PROYECTO FIN DE GRADO

GRADO EN INGENIERÍA INFROMÁTICA

CLIENTE TWITTER CON COMPRESIÓN DE DATOS

Autor: Plácido Fernández Declara

Tutores: Luis Miguel Sánchez García

Rafael Sotomayor Fernández

Colmenarejo, Febrero de 2013

Universidad Carlos III de Madrid
www.uc3m.es

RESUMEN

La popular red social Twitter es usada hoy en día en diversos lugares y momentos, gracias también a las redes móviles que nos proporcionan conexión a internet en casi cualquier lugar. Twitter permite publicar y mandar mensajes cortos de texto plano con un máximo de 140 caracteres, así como la lectura de estos mensajes cortos de otros usuarios a los que podemos seguir para tal propósito.

Hacer uso de este servicio en zonas con buena cobertura, o cuando usamos Wi-fi no supone generalmente ningún problema. Es cuando nos encontramos en zonas de poca cobertura cuando vemos problemas para hacer uso de este servicio, ya que las velocidades de descarga pueden disminuir hasta el punto de necesitar decenas de segundos para disponer de los mensajes cortos, los “Tweets”. A esto hay que añadir que los planes de datos actuales están limitados de alguna forma (al alcanzar un número de Mb), por lo que la cantidad de Mb usada es hoy también un factor importante.

Para ello esta aplicación ofrece la experiencia de un cliente Twitter sencillo en el que consultar los “Tweets” de a quien seguimos, así como publicar “Tweets”, haciendo uso de un Proxy que comprime estos datos considerablemente de forma que se reducen los tiempos de descarga para las zonas de poca cobertura y además se reduce el número de Mb consumidos por el dispositivo móvil.

Universidad Carlos III de Madrid
www.uc3m.es

Índice de contenido

Índice de Tablas

Índice de ilustraciones

1. INTRODUCCIÓN

En esta sección inicial se expone la visión general de este Proyecto Fin de Grado, cuál ha sido la motivación para realizarlo, qué necesidad tiene, por qué puede resultar útil y por último qué objetivos pretenden alcanzarse.

1.1 Motivación

El uso de los Smartphones está hoy muy extendido por todo el mundo, tanto que ya superan en número a los PCs convencionales y su uso va desde las zonas metropolitanas más pobladas hasta zonas rurales de unos pocos habitantes [1]. Y con estos están también extendiéndose las redes sociales como Facebook, Twitter, LinkedIn por mencionar algunas de las más populares. El uso de estas redes ha pasado de ser casi exclusivo en un computador a ser usado por millones de personas a través de los Smartphones, lo que permite su uso en las situaciones más diversas así como en cualquier lugar imaginado.

Esto da lugar a situaciones en las que la cobertura para datos es muy débil, o no tenemos disponible una conexión 3G (UMTS) o HSDPA [2] que permiten buenas velocidades de subida y descarga de datos. En muchos lugares las redes disponibles son GPRS, que permiten una velocidad (en el mejor de los casos) de 0,08 Mbps.

Ilustración 1 Twitter

La red social **Twitter**, que se aborda en este Proyecto Fin de Grado, descarga grupos de Tweets que contienen no sólo el propio mensaje corto de 140 caracteres si no otra información como el nombre de usuario, estado del Tweet, Retweets, posición del Tweet, y otra información similar. Además cada usuario que asociado a sus Tweets tiene asociado un avatar, una imagen que el móvil necesitará descargar. La descarga de todos esos Bytes en las situaciones adversas expuestas, puede suponer una espera prolongada o incluso que no llegue a completarse la descarga de dichos datos.

Este proyecto mejora esta situación de varias formas:

- Haciendo uso de un Proxy como servidor intermedio que se encarga de comprimir todos eso datos, reduciendo así el número de Bytes que hay descargar considerablemente y por tanto reduciendo el tiempo necesario para obtener los Tweets.
- Almacenando las imágenes de los avatares descargadas tanto en memoria principal como en secundaria para optimizar al máximo la reutilización de estas imágenes a modo de caché.
- Diseñando y ofreciendo varias interfaces y opciones para que el usuario pueda seleccionar el caso más óptimo en función de sus necesidades.

Ya que el número de usuarios de Twitter no para de crecer [3] y los Smartphones se extienden rápidamente por más países, la necesidad de mejorar las conexiones de este tipo de aplicaciones cobra cada vez más importancia. Las aplicaciones en las que puede tener una gran importancia mejorara la descarga de datos en velocidad y tamaño van desde la comodidad por los retardos al usar la aplicación hasta utilidad a la hora de comunicarse con una persona en riesgo como es el caso de una persona tratando de comunicarse desde una zona montañosa hasta informarse ante una catástrofe [4].

En un futuro idílico en el que la cobertura llegue sin problemas a todos, las conexiones no estén limitadas y sean rápidas, esta aplicación probablemente dejará de tener sentido salvo su aplicación en descargas de mucha información. Sin embargo ese momento es hoy por hoy lejano, mientras que la utilidad de este Proxy actualmente, es evidente.

1.2 Objetivos

El objetivo principal del proyecto es **el análisis, el diseño y el desarrollo de un sistema de compresión de datos para la red social Twitter desde un dispositivo móvil para ahorrar datos descargados, tanto para situaciones de mala cobertura como para situaciones en las que se desea utilizar menos datos**.

El sistema estará compuesto por dos subsistemas:

- El primer subsistema será una aplicación móvil para el Sistema Operativo Android que permita utilizar de manera básica la red social Twitter, permitiendo leer nuestro

Timeline y actualizar nuestro estado, optimizando la descarga de datos que se produce en la aplicación.

- El segundo subsistema será un servidor que actuará de Proxy comprimiendo los datos solicitados a la red social Twitter. El Proxy actuará de forma totalmente transparente al usuario, que tan solo interactuará desde la aplicación móvil.

Para lograr el objetivo se debe además alcanzar unos subobjetivos:

- Conocer los principios del lenguaje de programación Java para Android.
- Analizar y elegir el algoritmo de compresión más eficiente para los grupos de Tweets.
- Diseñar y desarrollar un servidor que actúe de Proxy comprimiendo los datos.
- Diseñar y desarrollar una aplicación móvil.
- Diseñar y desarrollar un sistema que cachee las imágenes de manera optimizada.
- Diseñar e implementar la comunicación entre los subsistemas.

1.3 Estructura del documento

En este apartado se detallan los diferentes capítulos que contiene este documento y en qué consisten:

- **Capítulo 1, Introducción**: Aporta la idea general del proyecto, con la motivación para realizarlo y los objetivos fundamentales que pretenden alcanzarse.
- **Capítulo 2, Estado del Arte**: Se explica una visión general de la tecnología utilizada para la realización del proyecto. En este capítulo se analizan distintos sistemas operativos para dispositivos móviles, distintos servidores para el Proxy, servicios Web y por último la red social Twitter.
- **Capítulo 3, Algoritmos de compresión**: Explicación, análisis y estudio de distintos algoritmos de compresión.
- **Capítulo 4, Análisis**: Necesidades básicas que el sistema debe satisfacer para desempeñar correctamente las funciones. Se analizarán los casos de uso y los requisitos software.
- **Capítulo 5, Diseño**: En este capítulo se procede a diseñar el sistema utilizando como base el capítulo anterior. Aquí se diseñarán las estructuras de datos, las comunicaciones y servicios necesarios y se diseñará la interfaz de usuario.
- **Capítulo 6, Desarrollo**: Se desarrollará el servidor Web y la aplicación con su interfaz.
- **Capítulo 7, Pruebas de evaluación**: Se realizarán y analizarán las pruebas de evaluación sobre el sistema desarrollado.
- **Capítulo 8, Conclusiones y líneas futuras**: Breve resolución del proyecto, las ideas extraídas de él y posibles vías de expansión.
- **Capítulo 9, Presupuesto**: Coste de desempeño y ejecución del proyecto.
- **Bibliografía**.
- **Manuales:** Manual de instalación del proyecto y manual de usuario.

1.4 Glosario de términos y acrónimos

Tabla 1 Términos

Término	Descripción
Twitter	Red social de mensajes cortos.
Tweet	Mensaje corto de hasta 140 caracteres.
Timeline	Lista de mensajes de los usuarios a los que se sigue.
Retweet	Publicar mensaje de otro usuario en Timeline.
Smartphone	Teléfono inteligente.
Android	SSOO de Google para dispositivos móviles.
iOS	iPhone Operating System, SSOO de Apple para dispositivos móviles.
iPhone	Smartphone de Apple.
iPad	Tablet de Apple.
iPod	Reproductor multimedia de Apple.

Tabla 2 Acrónimos

Acrónimo	Descripción
3G	Tercera generación de transmisión de voz y datos a través de telefonía móvil mediante UMTS.
UMTS	Universal Mobile Telecomunications System.
HSDPA	High Speed Downloadlink Packet Access.
GPRS	General Packet Radio Service.
Mbps	Mega bits por segundo
SSOO	Sistema Operativo
CDMA	Code Division Multiple Access
Bluetooth HDP	Bluetooth Health Device Profile
XAML	eXtensible Application Markup Language
SDK	Software Development Kit
RIM	Research in Motion
REST	Representational State Transfer
SOAP	Simple Object Access Protocol
URI	Uniform Resource Identifier
NDK	Native Development Kit

2. ESTADO DEL ARTE

En esta sección se aportará una visión global de las tecnologías usadas para el desarrollo de este Proyecto Fin de Grado, así como posibles alternativas.

2.1 Sistemas operativos móviles

En esta sección se expondrán la historia y las características de los sistemas operativos más destacados para teléfonos móviles inteligentes.

Los sistemas operativos que se analizarán serán:

- iOS
- Android
- Windows Phone
- Blackberry OS

2.1.1 iOS

Ilustración 2 iOS

El sistema operativo de Apple para sus dispositivos móviles como son el iPhone, iPad, iPod y Apple TV fue bautizado como iPhone Operating System por ser el primer dispositivo que lo adoptó y actualmente se conoce como iOS.

2.1.1.1 Historia

El iPhone fue el primer dispositivo que albergaría este sistema operativo y fue presentado en 2007 como el primer teléfono inteligente fabricado por Apple. Este sistema operativo se iría implementando posteriormente en otros dispositivos como el iPod Touch, ese mismo año.

Las versiones sucesivas de iOS se presentaron con cada nueva versión del iPhone, donde no fue hasta 2010 con el iPhone 4 donde el sistema operativo se bautizaría como iOS, siendo previamente conocido como iPhone Operating System.

Por último iOS sería el sistema operativo incluido en el Tablet iPad que Apple presentó en 2010, y las sucesivas versiones se han actualizado con el lanzamiento de nuevas versiones del iPhone siendo la última iOS 6 correspondiéndose con el iPhone 5 [5].

Ilustración 3 iPhone 5

2.1.1.2 Características

Las aplicaciones para los dispositivos iOS están desarrolladas en el lenguaje de programación Objetive-C, un lenguaje de programación orientado a objetos creado como un superconjunto de C que compila para la arquitectura ARM incluida en los dispositivos que utilizan iOS.

Los **desarrolladores** hacen uso del IDE Xcode que les permitiría desarrollaras las aplicaciones junto con el SDK que Apple proporciona. La utilización de este SDK junto con Xcode es gratuita, pero se deben tener en cuenta las siguientes limitaciones: sólo se puede usar en el sistema operativo MacOS para lo que es necesario poseer un Mac y para poder publicar la aplicación es necesario pagar al menos 99$ al año. Por otro lado los desarrolladores cuentan con la ventaja de desarrollar un software orientado a muy pocos dispositivos con características muy similares, por lo que el software está muy optimizado para el hardware sobre el que se ejecuta.

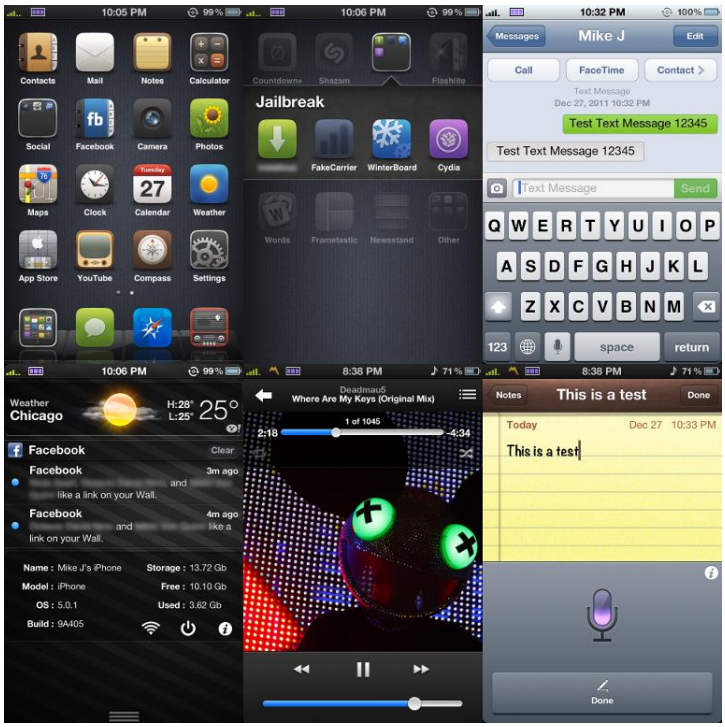

Ilustración 4 Interfaz iOS

La **interfaz** de iOS cuenta con una pantalla principal que alberga los iconos de las aplicaciones instaladas en el dispositivo y que se desliza lateralmente. En general iOS solo puede ejecutar una aplicación a la vez, donde la multitarea estaba reservada a aplicaciones del sistema y no fue hasta iOS 4 cuando Apple liberó una API para que los desarrolladores pudiesen implementar la multitarea.

Además todos los dispositivos iOS cuentan con cámara, bluetooth, WI-FI y GPS. Algunos dispositivos como el iPhone y algunas versiones del iPad cuentan además con conexión 3G para disponer de conexión a internet en cualquier lugar con cobertura para ello.

Por último, la **arquitectura** de iOS se compone de 3 capas, como se puede ver en la Ilustración 5 Arquitectura iOS. En el primer nivel encontramos la capa de aplicaciones, conocida como Cocoa Touch, una API del sistema operativo que proporciona una capa de abstracción para crear aplicaciones. Después encontramos la capa Core Service donde se encuentran las API de seguridad del sistema basadas en los servicios de la última capa, la Core OS, la capa del núcleo del sistema.

Ilustración 5 Arquitectura iOS

2.1.2 Android

Ilustración 6 Android

El sistema Android analizado a continuación es el sistema elegido para el desarrollo de la aplicación móvil del proyecto.

2.1.2.1 Historia

Ilustración 7 Nexus

Android fue fundada como Android Inc. para ser comprada en 2005 por Google y siguió siendo desarrollado por la Open Handset Alliance, unión de fabricantes de software y hardware como T-Mobile, HTC o Qualcomm, liderado por Google. Android está basado en Linux y está destinado a dispositivos móviles.

El SDK de Android se presentó en 2007, pero no fue hasta pasado un año cuando vio la luz el primer Smartphone con Andorid, un HTC, con Android 1.0 que incluía los servicios de Google.

En 2009 se actualizó Android a 1.5 donde se inició la lista de nombres de postres que sigue en cada nueva versión de Android, siendo esta primera llamada Cupcake. Esta versión mejoraría la velocidad general del sistema, la localización GPS y mayor integración con servicios como YouTube y Picassa y además contaría con teclado virtual.

Más tarde en el mismo año saldría la versión 1.6 Donut, que incluiría la búsqueda escrita y por voz, un indicador de uso de batería y daría soporte para CDMA.

Casi enseguida, un mes después, se lanzó otra nueva versión, la 2.0 Eclair corrigiendo más fallos de la versión anterior. En esta versión se añadirían sincronización de contactos, soporte para Bluetooth 2.1, una nueva interfaz para el navegador con soporte para HTML 5 y el calendario se ampliaría.

En 2010 se liberaron Android 2.1 y 2.2 Froyo, que incluiría Adobe Flash 10.1, teclado en varios idiomas y soporte para punto de acceso.

A finales de 2010 salió la hasta ahora versión más popular de Android, la 2.3 Gingerbread. Esta versión incorporaría llamadas a través de internet, NFC, teclado nuevo y nuevas formas de copiado y pegado de texto.

La siguiente revolución de Android fue el soporte para Tablets que incluiría en la versión 3.0 Honeycomb, en ese momento versión solo para tablets.

En octubre 2011 Google presentó la renovación más importante de Google en la que Android alcanzaría la madurez como Sistema Operativo. Esta versión unificaría la versión de tablets y móviles con interfaz optimizada. Se incluiría una nueva tipografía llamada Roboto, una nueva forma de desbloqueo mediante lectura de gestos de la cara, soporte para Bluetooth HDP y Wi-Fi Direct.

En el Google I/O de 2012, en junio, se presentó la versión 4.1 Jelly Bean. Esta nueva versión sería pionera en el primer Tablet de Google, el Nexus 7, fabricado por Asus. Jelly Bean mejoraría la fluidez general del sistema con el conocido como Project Butter. Además mejoraría el rendimiento y la compatibilidad de HTML 5 con el navegador.

La última versión liberada se considera una actualización de Jelly Bean y su versionado es 4.2. Junto con esta versión se presenta el más reciente teléfono de Google, el Nexus 4.

Esta distribución de versiones es hoy por hoy uno de los mayores problemas de Android, ya que está bastante fragmentado, con muchos dispositivos en el mercado con versiones muy distintas en los dispositivos.

2.1.2.2 Características

Android es considerado una plataforma móvil donde está incluido el sistema operativo móvil basado en Linux, junto con unas aplicaciones para interactuar con el mismo. El **desarrollo de aplicaciones** móviles para Android se realiza mediante el lenguaje de programación Java junto con el SDK proporcionado por Google, donde además podemos utilizar las librerías del NDK para programar en C.

Una característica particular de Android es que funciona sobre una máquina virtual propia conocida como Dalvik, la cual es similar a la JVM pero está optimizada para el funcionamiento en dispositivos móviles. Además Android ofrece varias características importantes como un navegador integrado en el sistema operativo basado en Webkit, gráficos optimizados por OpenGL, bases de datos SQLite para que los desarrolladores puedan almacenar la información y soporte para los habituales formatos de audio, video e imagen.

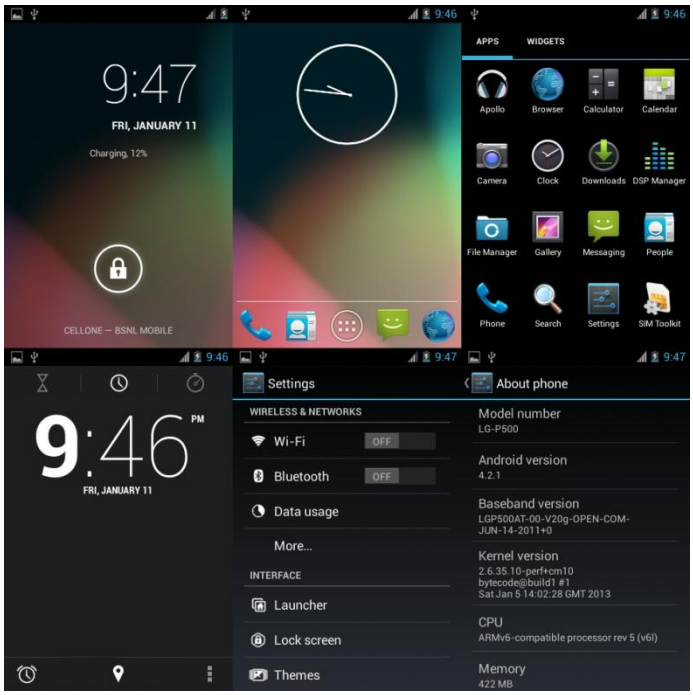

Ilustración 8 Interfaz Android

Las conexiones y sensores dependen del dispositivo en cuestión, donde se puede encontrar una amplia variedad.

La arquitectura cuenta con 5 componentes como se puede ver en la Ilustración 8 Interfaz Android.

- **Linux Kernel**: Android está basado en el kernel de Linux y comparte con este la gestión de memoria, procesos o servicios básicos de seguridad entre otros. Su funcionamiento está basado en el kernel y este actúa como una capa entre el hardware y el resto del software.
- **Libraries**: Las bibliotecas incluidas de C/C++ son utilizadas por diversas partes del sistema operativo. Estas son ofrecidas a los desarrolladores a través del marco de aplicaciones.
- **Android Runtime**: También se tratan de librerías ofrecidas a los desarrolladores, pero estas son las que ofrecen la mayoría de funcionalidades disponibles en las bibliotecas de Java.
- **Application Framework**: El marco de aplicaciones es la capa disponible para los desarrolladores que les permiten crear las aplicaciones que podemos encontrar para este sistema operativo. Los desarrolladores tienen acceso total a la API que tiene una arquitectura diseñada para la reutilización de componentes.
- **Applications**: La última capa son las aplicaciones. Algunas de ellas vienen incluidas en el sistema operativo para poder interaccionar con el mismo, y son aplicaciones como un calendario, el acceso a los SMS, mapas, correo electrónico, etc...

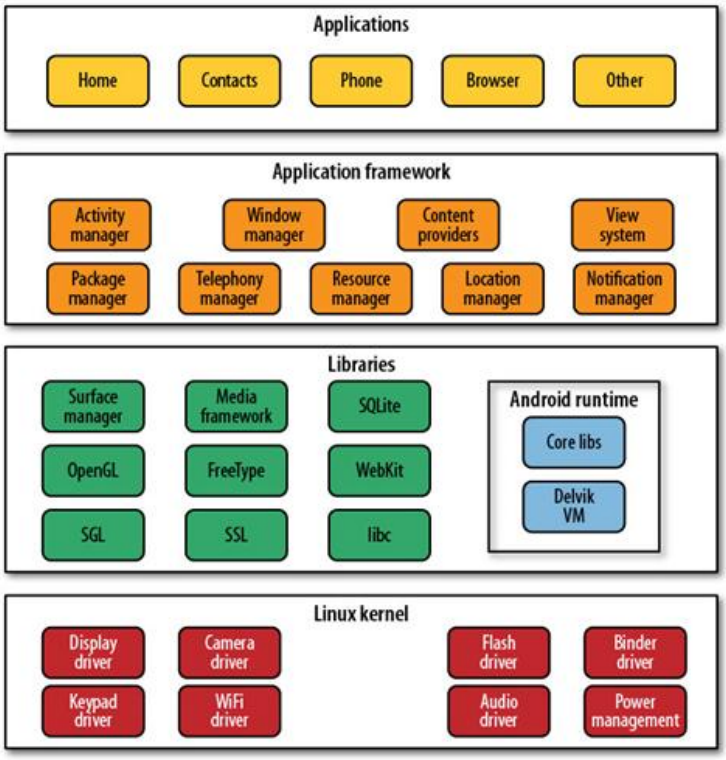

Ilustración 9 Arquitectura Android

2.1.2.3 Componentes de Android

Ya que Android es el sistema operativo elegido para este proyecto profundizaremos en los componentes que se usarán en la aplicación, conocidos como "**Main Building Blocks**". Estos componentes son usados a lo largo del desarrollo de la aplicación y son descritos para ayudar a comprender mejor el diseño y el desarrollo de la aplicación. [6]

2.1.2.3.1 Activities

Los Activity son el componente más usado en Android, son las pantallas visibles para el usuario, y solo puede ver un Activity al mismo tiempo.

Una buena forma de explicar lo que es un Activity es una página web. Una web está compuesta por varias páginas y posee una "home page" de la misma forma que una aplicación de Android tiene una "main" Activity o actividad principal. Igual que en una web podemos saltar de una página a otra y el usuario sólo puede ver una a la vez, un usuario puede saltar en una App de un Activity a otro.

Lanzar un Activity es una operación que puede ser costosa ya que requiere de crear un nuevo proceso, reservar memoria para los objetos de la interfaz, "inflar" los XML que definen esta

interfaz y por último pintar el Activity para que sea visible al usuario. Para evitar que constantemente se estén lanzando Activities desde cero Android cuenta con el Activity Manager.

El Activity Manager se encarga de crear, destruir, pausar, reanudar y parar las Activities. Por ejemplo cuando un usuario lanza una App por primera vez, el Activity Manager trae el Activity a la pantalla. Después cuando el usuario cambia a otra pantalla, el Activity Manager guardará ese Activity en memoria para cuando el usuario vuelva al primer Activity tarde mucho menos en lanzarlo. Así, los Activities que no se usan en un tiempo son destruidos automáticamente para dejar hueco a nuevos Activities.

En la imagen se puede apreciar los estados del Activity y como se puede pasar de un método a otro.

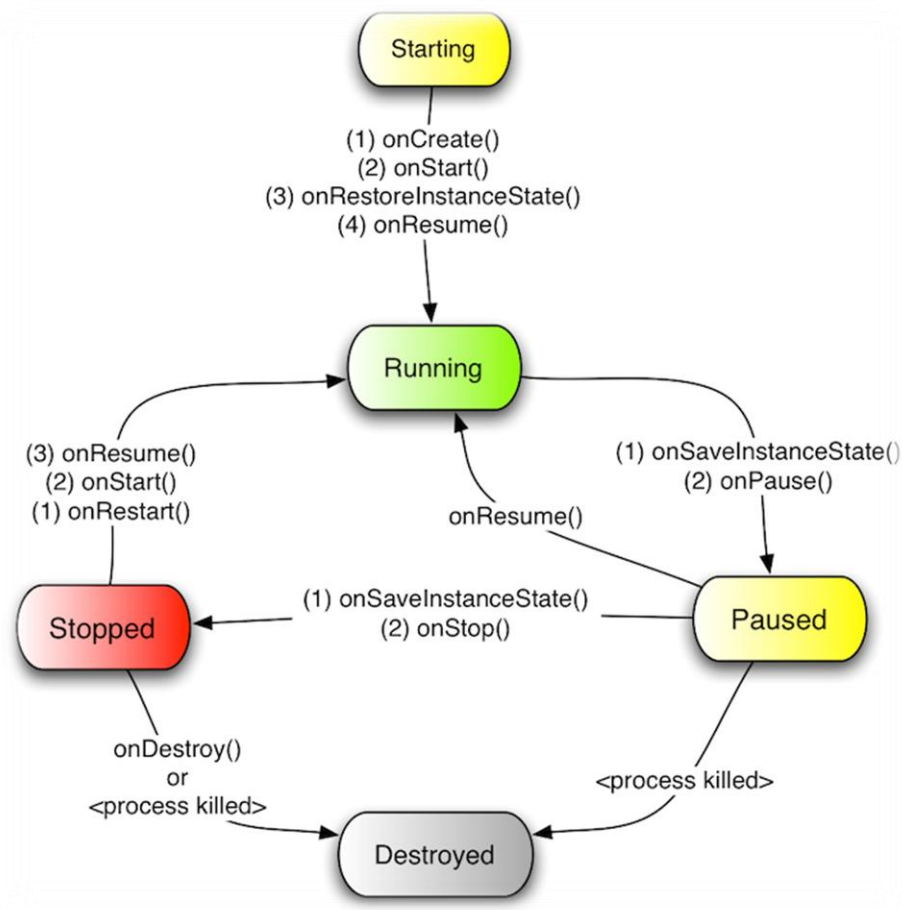

Ilustración 10 Activity

2.1.2.3.2 Intents

Los Intents son mensajes que son mandados entre el resto de "Building Blocks" o componentes de Android. A través de estos se pueden lanzar nuevos Activities, pueden lanzar

un Service o pararlo o pueden actuar como mensajeros en general o “broadcasts”. Se debe tener en cuenta además que los Intents son asíncronos, lo que significa que el código que están mandando no tiene que esperar al resto para ser completados.

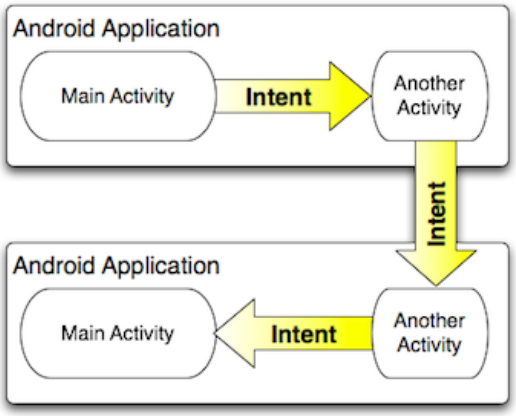

Ilustración 11 Intents

2.1.2.3.3 Services

Junto con los Activities el otro gran componente de Android son los Services. Estos se ejecutan en segundo plano y no tienen ningún tipo de interfaz para el usuario. Son útiles para acciones que van a tardar en ejecutarse independientemente de lo que ocurra en la interfaz o simplemente si queremos ejecutar algún proceso mientras nos movemos entre distintas Apps.

Estos Services no deben ser confundidos con los servicios o demonios de Linux, que se tratan de componentes de mucho más bajo nivel en el sistema operativo.

El ciclo de vida de un Service es muy sencillo como se puede ver en la imagen y debe ser controlado por el desarrollador para evitar que un servicio se ejecute demasiado tiempo y así no consumir recursos innecesarios de CPU y batería.

Ilustración 12 Service

2.1.2.3.4 Content Providers

Aunque no es tan usado como los otros componentes, los Content Providers juegan un papel importante en la aplicación de este proyecto ya que se encargan de manejar los Tweets.

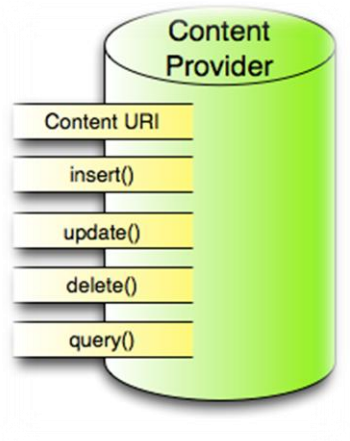

Ilustración 13 Content Provider

Estos son interfaces para intercambiar datos entre las aplicaciones o para manejar grandes cantidades de datos dentro de una aplicación. Mientras que con los Intents podemos pasar una pequeña cantidad de información entre los componentes, cuando queremos pasar mucha información o manejar muchos datos constantemente se deben usar los Content Providers.

Estos son usados constantemente en las Apps de Android, por ejemplo la aplicación de los contactos hace uso de un Content Provider para poder ofrecer los contactos a cualquier App que los solicite.

Los métodos asociados a los Content Providers son relativamente simples y muy parecidos al uso que daríamos a una BBDD.

2.1.2.3.5 Application Context

Por último y aunque no se trata de un "Main Building Block" es importante explicar lo que hace un Application Context. El conjunto del resto de componentes forman una aplicación, y estos viven en el mismo contexto de la aplicación que es el Application Context.

Este se refiere al entorno de la aplicación y recoge los procesos y variables que en esta se ejecutan, permitiendo reunión toda la información y recursos para ser compartidos entre el resto de componentes.

De esta forma desde cualquier punto de la App en el que estemos podemos llamar a getApplicationContext() para recuperar todos los datos asociados a esta. Esto es usado constantemente a lo largo del proyecto.

2.1.3 Windows Phone

Ilustración 14 WIndows Phone 8

Windows Phone es el sistema operativo para Smartphones de Microsoft, que se introdujo en el mercado de los dispositivos móviles algo más tarde que el resto de competidores pero que consigue hacerse un hueco en el mercado de este tipo de dispositivos.

2.1.3.1 Historia

Ilustración 15 Windows Phone 8 HTC

Windows Phone se trata del sucesor del anterior Windows Mobile, un sistema operativo que fue diseñado para Smartphones como una ramificación de Windows CE y que originalmente fue conocido como Pocket PC.

Este fue presentado en el año 2000 y el los posteriores años fue integrándose con los servicios de Microsoft como el MSN Messenger, el Media Player, Internet Explorer u Office Mobile. No fue conocido como Windows Mobile hasta el año 2003 e introdujo soporte Multitouch ya en el año 2009.

Pero la verdadera apuesta por hacerse un hueco en un mercado liderado por Android y iOS fue en 2010 cuando Microsoft presentó el Windows Phone 7, que posteriormente en el año 2012 evolucionaría a la versión 8 para alcanzar una madurez y una calidad a la altura de sus competidores en el sector. [7]

2.1.3.2 Características

El núcleo del sistema operativo de Windows Phone está basado en el Windows Embedded CE y para desarrollar las aplicaciones se puede usar Silverlight o XNA Framework.

Silverlight permite desarrollar aplicaciones ricas en efectos visuales y transiciones basadas en XAML. Dentro de Silverlight se incluye el Microsoft .NET Compact Framework que se trata de un subconjunto de las conocidas librerías .NET Framework. Los desarrolladores pueden descargar e instalar herramientas gratuitas y el SDK para desarrollar aplicaciones, pero además se pueden publicar hasta 100 Apps de forma gratuita. [8]

Por otra parte XNA se trata de una implementación nativa de .NET Compact Framework donde están incluidas un amplio conjunto de bibliotecas orientadas al desarrollo de videojuegos como por ejemplo el tratamiento de sonidos y videos, carga de modelos y texturas, etc.

Windows Phone como las otras plataformas analizadas está pensada para dispositivos táctiles, sin embargo la **interfaz** de Windows Phone es innovadora y trata de diferenciarse de sus competidores iOS y Android con una pantalla de inicio compuesta por “Live Tiles”, unos mosaicos dinámicos que enlazan con las aplicaciones, funciones o características mostrando información útil de cada una en función del usuario. [9]

El sistema operativo utiliza el navegador Internet Explorer Mobile 9 basado en Internet Explorer 9 y para las búsquedas hace uso del motor de búsqueda de Microsoft: Bing. Además el sistema operativo está muy bien integrado con la suite ofimática Office de Microsoft y tiene pleno acceso al servicio de almacenamiento en la nube SkyDrive para compartir los archivos.

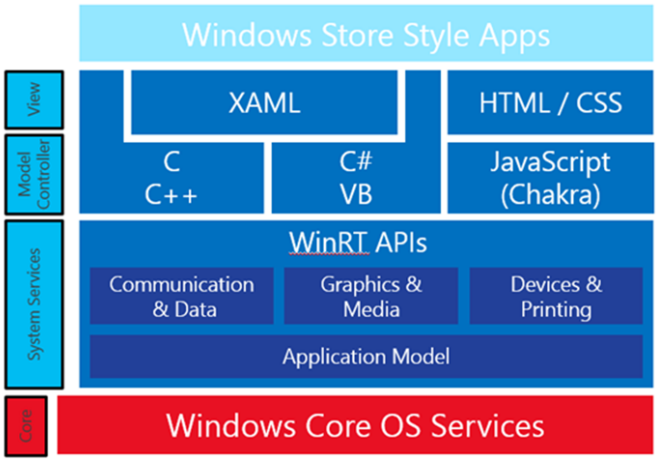

Ilustración 16 Arquitectura Windows Phone

Para completar la integración del sistema operativo con los servicio de Microsoft se incluyen funcionalidades de Xbox Live para completar "Achievements" o retar a amigos.

La **arquitectura** se puede observar en la ilustración, con un framework para desarrollar las apps del sistema y controladores para los distintos componentes hardware que pueda tener el dispositivo ya que Windows Phone a diferencia de iOS está enfocado a una mayor variedad de dispositivos de distintos fabricantes. [10]

2.1.4 Blackberry OS

Research in Motion (RIM) se trata de la empresa canadiense detrás del sistema operativo Blackberry OS para dispositivos móviles. Estos dispositivos son además fabricados por la propia empresa integrando de esta forma hardware y software en la misma empresa.

Ilustración 17 Blackberry

2.1.4.1 Historia

El desarrollo comenzó en 1999 con la aparición de los primeros "handheld", que permitían el acceso a correo electrónico, navegación web y sincronización como cuentas de Exchange o Lotus Notes aparte de poder usar el dispositivo como un teléfono móvil.

Cuando realmente RIM comenzó a conseguir el éxito que vendría después fue con sus terminales de la serie 5000 y 6000. Fue con la serie 7200 de sus dispositivos la primera en incluir características como pantalla a color, con la 7250 llego el Bluetooth, después la Wi-fi con la 7270 y el GPS con la 7520. La expansión de RIM llegó hasta los 5 millones de usuarios en 2006 con la serie 8700.

En el año 2010 con iOS y Android compitiendo de manera directa Blackberry lanzó la versión 6 de sus sistema operativo donde mejorarían notablemente los aspectos multimedia enfocados al usuario, la integración con redes sociales y su servicio estrella, la mensajería instantánea.

RIM lo intentó también en el mundo de los tablets lanzando la primera en 2011, la PlayBook, donde modificaron ligeramente el sistema operativo para que se adaptase a las pantallas más grandes y se denominó Blackberry Tablet OS. Este está basado en QNX Neutrino, un sistema operativo de tiempo real derivado de UNIX. QNX probablemente será incluido en el próximo sistema operativo de Blackberry, el Blackberry 10 OS. [11]

2.1.4.2 Características

El sistema operativo de Blackberry está orientado al entorno profesional por su gestión del correo electrónico, la agenda y la seguridad de su sistema de mensajería. Además una característica identificativa de Blackberry son sus dispositivos con teclado físico QWERTY, presente en la mayoría de sus dispositivos. Sin embargo es muy posible que está característica desaparezca en los nuevos dispositivos que Blackberry pretende lanzar junto con su versión 10 de su sistema operativo.

La seguridad ofrecida por Blackberry depende de suscripciones a los servicios asociados de mensajería instantánea y gestión de correo. [12] El servicio de seguridad asociado para empresas, profesionales y usuarios que lo deseen conocido como Blackberry Enterprise Server (BES) cuenta con control total del buzón de entrada e identificación de correos electrónicos, gestión de filtros, comprime y optimiza los mensajes encriptándolos y enviándolos a los dispositivos asociados a la cuenta. El servicio para usuarios particulares, Blackberry Internet Service (BIS) permite el uso de la tecnología PUSH y puede sincronizar hasta un máximo de 10 cuentas de correo electrónico.

La **arquitectura** de Blackberry OS dispone de un sistema operativo escrito en C++ con características multitarea, comunicación entre procesos, hilos, soporte para dispositivos de entrada.

Ilustración 18 Blackberry 10

2.2 Servicios Web

Los servicios Web son tecnologías software que usan protocolos y estándares para intercambiar información entre las aplicaciones, lo que nos permite comunicar dos dispositivos conectados a una red, en nuestro caso un dispositivo móvil con un servidor.

Los servicios Web permiten comunicación entre aplicaciones desarrolladas en lenguajes de programación distintos y ejecutados sobre cualquier plataforma, lo que permite la expansión de un desarrollo inicial a diferentes tecnologías y dispositivos.

Para este proyecto se analizan REST y SOAP y además se compararán.

2.2.1 REST

La Transferencia de Estado Representacional (Representational State Transfer) o REST es una técnica de arquitectura software para sistemas distribuidos. El término nació en el año 2000 en la tesis doctoral de Roy Fienlding, uno de los autores de la especificación del protocolo HTTP.

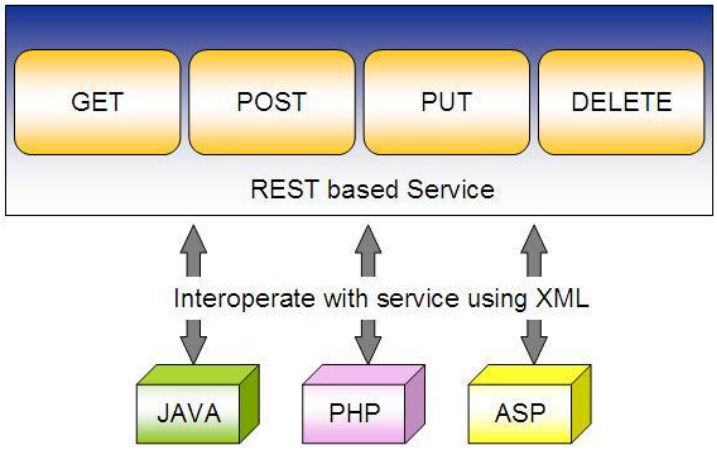

Ilustración 19 REST

Originalmente REST se refería a un conjunto de principios de arquitectura, pero hoy en día es usado para describir cualquier interfaz que transmite datos específicos de un dominio HTTP sin una capa adicional como hace SOAP. Cabe desatacar que REST no se trata de un estándar sino que es un conjunto de principios de arquitectura basado en estándares tales como HTML, HTTP o URL entre otros. [13]

Los principios de REST son los siguientes:

- **Asignación de un ID a los recursos**: En el caso de REST se trata de asignación de URIs, que comprenden un espacio de nombres global que permite identificar cada recurso de manera única.

- **Hipermedia como motor de estado aplicativo**: Este principio puede resumirse como que los recursos deben ser vinculados, tanto para el usuario en forma de los típicos links que podemos encontrar en la web como de cara al servidor, que pueda acceder a los recursos por medio de vínculos. Esto convierte la aplicación en una aplicación dinámica que puede cambiar su estado por medio de vínculos.
- **Utilización de los métodos estándar**: Esto permite que haciendo uso de las URIs y los vínculos la aplicación sepa qué hacer con los recursos gracias al uso de una misma interfaz y un mismo conjunto de métodos. Estos métodos son los conocidos GET y POST, pero además incluye los métodos PUT, DELETE, HEAD y OPTIONS que están definidos en la especificación HTTP.
- **Recursos con múltiples representaciones**: las ventajas de ofrecer varias representaciones de los recursos (por ejemplo HTML y XML) son varias, como que los recursos puedan ser consumidos no solo por nuestra aplicación, si no por cualquier navegador estándar o cualquiera que conozca cómo funciona la Web. Esto es porque no existe un formato ideal estandarizado que sea usado por todos, por lo que es necesario proveer de varias representaciones para poder interaccionar con distintos sistemas.
- **Comunicar sin un estado**: Esto no significa que una aplicación no pueda tener un estado, de hecho debe tenerlo. REST no guarda el estado de la comunicación de los clientes que se comunican con él más allá de la petición individual que le hace cada uno, todo ello para mejorar la escalabilidad, ya que el servidor no tiene que mantener el estado de un cliente, y así puede dar servicio a otro en caso de mucha demanda.

2.2.1.1 Restlet

Este framework es el elegido para el desarrollo del proyecto por ser muy completo y ligero, por hacer uso específico de las tecnologías usadas como son Java, el servidor Google App Engine analizado posteriormente y Android y por último por ser de código abierto.

Restlet fue creado en 2007 por Jerome Louvel para desarrollar clientes y servicios REST en la plataforma Java. Este cumple con los conceptos de estilo de arquitectura de REST, recursos, representación, conectores y componentes. [14]

Además soporta la mayoría de estándares de Internet, de datos y servicios como HTTP, HTTPS, SMTP, XML, JSON… pero además cuenta con multitud de extensiones para su integración con Spring, Jetty, Servlet…

Ilustración 20 Restlet

Existen cinco ediciones de Restlet y cada una de ellas está orientada a un entorno de desarrollo específico:

- GWT o Google Web Toolkit para aplicaciones AJAX que son desplegadas en navegadores de escritorio sin necesidad de plugins.
- GAE o Google App Engine para desplegar aplicaciones en la plataforma de computación en la nube de Google.
- Android para ser usado en los Smartphones compatibles con este sistema operativo.
- Java SE para aplicaciones que hacen uso de la plataforma JVM.
- Java EE para despliegues en Servlets de Java.

2.2.1.2 Apache Axis

Ya que REST es un estilo de arquitectura y no una tecnología específica o una especificación, servicios como Axis que se tratan de una implementación de SOAP permite crear servicios RESTful.

Apache Axis se trata de una implementación de SOAP (analizado posteriormente), que proporciona un entorno de ejecución para Servicios Web implementados en Java y C++. Haciendo uso de Apache Axis se pueden crear aplicaciones computacionales interoperables y distribuidas. [15]

El desarrollo se hace bajo la supervisión de la Apache Software Foundation.

2.2.1.3 Apache Axis 2

Esta versión se trata de una reimplementación desde cero de la versión analizada anteriormente. Esta creció de manera paralela e independiente de Apache Axis y es analizada por separado porque implementa una especificación distinta.

Mientras que Apache Axis implementa JAX-RPC, Axis 2 implementa la especificación JAX-WS del Java Comunity Process, WS-Messaging y WS-Security. Este está pensando para ampliar su funcionalidad en el futuro con plugins.

Axis2 promete las siguientes características: velocidad, uso reducido de memoria, AXIOM como modelo de objetos extensible y optimizado, despliegue instantáneo, servicios web asíncronos por medio de clientes y transportes no bloqueantes, soporte de MEP (Message Exchange Patterns), flexibilidad para insertar extensiones al motor, estabilidad, despliegue orientado a componentes, framework de transporte (senders y listeners por protocolos como

FTP, SMTP…), soporte para WSDL (Web Services Description Language), agregados, composición y extensibilidad.

2.2.2 SOAP

SOAP (Simple Object Access Protocol) si es un protocolo que nos permite comunicar aplicaciones por medio de la red intercambiando datos XML. El protocolo deriva de otro llamado XML-RPC, creado en 1998 por David Winer y fue desarrollado por empresas como Microsoft e IBM. SOAP es uno de los protocolos más extendidos en la web.

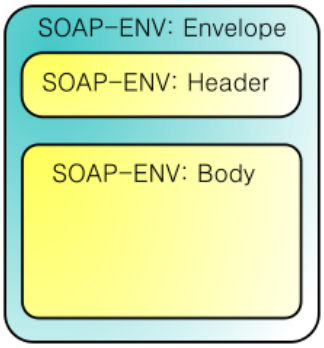

Ilustración 21 SOAP

SOAP, al estar basado en XML, es un protocolo independiente de la plataforma y el lenguaje lo que ha facilitado mucho su expansión en la web. Pero además SOAP intenta alcanzar el objetivo de convertirse en un estándar de invocación de servicios remotos usando HTTP para la transmisión y XML para la codificación de los datos.

Este protocolo tiene tres características principales:

- **Extensibilidad**: seguridad y WS-routing son extensiones aplicadas en el desarrollo.
- **Neutralidad**: SOAP se puede usar sobre cualquier protocolo de transporte como HTTP, TCP y SMTP.
- **Independencia**: SOAP permite cualquier modelo de programación.

Y se compone de tres partes:

- Un envoltorio que define el contenido del mensaje y cómo debe ser procesado.
- Un conjunto de reglas de codificación para expresar instancias de tipos de datos.
- Una conversión para representar las llamadas a procedimientos y respuestas.

HTTP es el protocolo usado para la conexión en internet, por ello al usarlo en SOAP se asegura que los clientes con navegadores estandarizados pueden conectarse a un servidor utilizando esta tecnología. Por otro lado XML también se trata de un estándar para el intercambio de datos y al ser usado para codificar los datos nos encontramos con muchos menos problemas de incompatibilidades que otros protocolos. [16]

2.2.3 Comparativa

Una vez analizados ambos los comparamos para poder elegir uno u otro correctamente.

Ventajas de SOAP:

- El lenguaje WSDL nos permite definir con mucho detalle un contrato en el que se describen todas las funciones de una interfaz.
- SI vamos a utilizar una arquitectura que tenga requerimientos complejos y no funcionales SOAP se comportará mejor. Es el caso de transacciones, seguridad o direccionamiento. Esto es porque SOAP mantiene la información contextual y el estado de la conexión.
- SOAP es conveniente si nuestra arquitectura maneja procesos de manera asíncrona por el tiempo que necesita para realizar una parte del procesado de la petición.

Ventajas de REST:

- Cuando un servicio web no requiere de tener estado ya que REST no lo guarda.
- Nos permite mejorar el rendimiento añadiendo una infraestructura de caching.
- REST es conveniente si en nuestra aplicación tanto cliente como servidor conocen el contexto y contenido que van a intercambiar.
- REST es mejor para consumir servicios web en smartphones con recursos limitados.
- Si vamos a crear un servicio que hace uso de otros sitios web existentes REST nos lo pondrá mucho más fácil.

Además cabe destacar algunas diferencias entre ellos:

- En REST la interacción con el usuario se realiza a través de formularios mientras que en SOAP es un flujo de eventos.
- REST hace uso de pocas operaciones pero con muchos recursos, pero en SOAP es al contrario, necesita muchas operaciones pero de pocos recursos.
- REST nos permite nombrar los recursos de una manera clara y consistente, las URI, SOAP por otra parte no tiene un mecanismo claro de nombrado.
- REST está más centrado en aplicaciones escalables de gran rendimiento para sistemas distribuidos, y SOAP está más centrado en el diseño de aplicaciones distribuidas.
- REST es síncrono y hace uso de HTTP y XML, mientras que SOAP puede ser síncrono o asíncrono y como codificación utiliza XML SChema.
- REST utiliza el Web Application Description Language (WADL) que no necesita de tipado fuerte. SOAP utiliza el Web Service Description Language (WSDL) que permite escribir con detalle un servicio Web y requiere un tipado fuerte.
- REST utiliza HTTPS y SOAP hace uso de WS-Security.

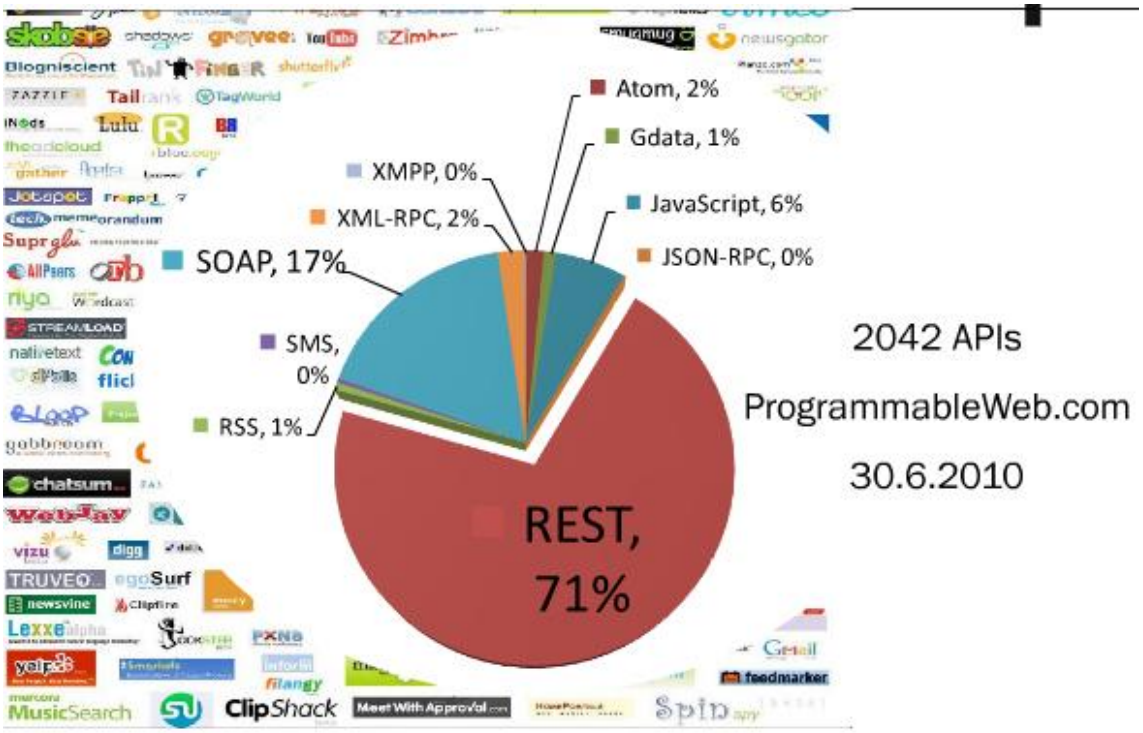

Ilustración 22 REST vs SOAP

REST es el protocolo elegido para mostrar cantidades ingentes de datos y son empresas como Google, eBay o Amazon que utilizan REST para algunos de sus servicios. [17]

2.3 Platform as a Service o Plataforma como servicio

En esta sección se expondrán las características principales de algunas plataformas de computación en la nube conocidas como **PaaS**. La plataforma como servicio se trata de un entorno contenido en una pila básica de sistemas, componentes o APIs pre configuradas y listas para integrarse sobre una tecnología como Java, Python o un sistema Linux.

Este tipo de servicio ofrece computación en la nube de altas prestaciones, gran escalabilidad y una impresionante facilidad para poner en marcha una aplicación en la nube que sea utilizada por miles de personas lo que lo convierte en el entorno perfecto para este proyecto.

La oferta de este tipo de servicios incluye a grandes empresas como Google con Google App Engine, Microsoft con Windows Azure, Red Hat con OpenShift o Amazon con Amazon Web Services entre muchos otros. [18]

2.3.1 Google App Engine

GAE o Google App Engine es la apuesta de Google en este tipo de servicios. Las aplicaciones desplegadas en GAE corren en procesos aislados (sandboxed) pero en multitud de servidores lo que hace que la escalabilidad de las aplicaciones sea más sencilla. Tanto es así que GAE ofrece escalabilidad automática para sus aplicaciones web de manera transparente al desarrollador. Cuando una aplicación web incrementa su número de peticiones, el GAE automáticamente dispone a la aplicación de más recursos de memoria y CPU para cubrir las demandas de la aplicación.

GAE soporta Python y Java y está en fase de pruebas el lenguaje de programación de Google: Go. Sin embargo al utilizar tecnologías basadas en máquinas virtuales como la JVM soporta por extensión otras tecnologías como Scala, JRuby, Jython o PHP entre otras.

Además de la facilidad que ofrece Google a la hora de desplegar una aplicación y escalarla, GAE puede presumir de que sus aplicaciones tienen una disponibilidad del 99,95%.

Por otra parte GAE tiene restricciones a la hora de usar este servicio, ya que está limitado a los lenguajes mencionados y además las APIs disponibles ofrecen guardar y recoger datos de una base de datos no relacional propia de Google, las peticiones deben ser HTTP, mandar e-mails, manipular imágenes y cachear. Lo que salga de estas características probablemente no correrá en GAE.

Ilustración 23 GAE

Para almacenar los datos GAE hace uso de los “datastore” que posee una sintaxis similar a SQL conocida como “GQL” y se trata de una base de datos no relacional.

Por último GAE es gratuito para los desarrolladores con la limitación de que al alcanzar determinados usos de CPU, memoria o almacenamiento Google cobrará proporcionalmente según vayamos necesitando más recursos.

2.3.2 Heroku

Heroku pertenece a Salesforce.com y es otro de los PaaS más usados siendo también una de las primeras plataformas de computación en la nube. Este soporta varios lenguajes como Ruby, Java, Node.js, Scala, Clojure, Python y PHP. El sistema operativo base que utiliza es Debian aunque en los últimos meses está usando Ubuntu.

Ilustración 24 Heroku

Heroku tiene un funcionamiento muy similar a Google App Engine, pero Heroku ofrece mayor libertad general ya que permite por ejemplo ejecutar hilos desde los servlets, conectarse a un servidor IMAP, etc… tiene la ventaja también de que si tienes un servidor escrito en Java ya funcionando es muy probable que lo puedas desplegar en Heroku y hacerlo funcionar sin modificarlo, cuando esto en GAE es muy probable que no ocurra.

Ademas Heroku ofrece otro tipo de almacenamiento de datos, en este caso haciendo uso de una base de datos SQL.

El precio de Heroku también es gratuito hasta cierto límite, con unas cuotas muy parecidas a las de GAE.

2.3.3 OpenShift

OpenShift se trata de la apuesta de Red Hat en la computación en la nube que cuenta además con una versión para una nube privada, OpenShift Enterprise.

OpenShift es de código abierto, con el nombre de OpenShift Origin y está disponible en GitHub para todo el mundo lo que lo hace muy atractivo para los desarrolladores así como para entender mejor como funciona y así poder construir la aplicación que queramos desplegar en este servicio de forma más optimizada.

Los lenguajes soportados son Node.js, Ruby, Python, PHP, Perl y Java, una oferta muy similar a Heroku. Sin embargo OpenShift soporta programas binarios que sean aplicaciones web siempre y cuando estas se puedan ejecutar en Red Hat Enterprise Linux, lo que amplía la oferta a muchos otros lenguajes y frameworks.

Ilustración 25 OpenShit PaaS

Para el almacenamiento de datos se pueden usar servicios conocidos como son MySQL, PostgresSQL y MongoDB que pueden cubrir varios tipos de necesidades según la aplicación.

2.3.4 Windows Azure

La apuesta de Microsoft por la computación en la nube pensada para construir, desplegar y manejar aplicaciones que corran sobre la red global de datacenters de Microsft recibe el nombre de Windows Azure.

Ilustración 26 WIndows Azure

Windows Azure ofrece servicios similares a los PaaS anteriormente analizados pero además ofrece la posibilidad de crear páginas web en .NET, crear máquinas virtuales que corren en los Microsoft Data Centers, entre las bases de datos podemos usar SQL y almacenamiento BLOB y podemos hacer uso de otras tecnologías como Windows Azure Service Bus.

Windows Azure a diferencia de los anteriores no tiene opciones gratuitas de uso, al menos a largo plazo, ya que lo que si nos permite Microsoft es probar el servicio con unas condiciones muy buenas respecto a los otros, pero durante 90 días, pasado ese periodo la opción deja de ser gratuita en cualquier caso.

2.4 Twitter

Aquí se analiza el servicio Twitter con su API y su funcionamiento más específicamente para comprender mejor como se aborda el proyecto.

Twitter se creó en 2006 como una pequeña start-up, al principio usado internamente pero lanzada al público ese mismo año. En 2008 Twitter estaba formado por 18 personas, durante 2009 se multiplicó por 4 su plantilla y sigue creciendo.

Ilustración 27 Twitter Logo

La interfaz de Twitter está escrita en Ruby on Rails, pero los mensajes se guardan en un servidor escrito usando el lenguaje de programación Scala. Twitter liberó una API para que los desarrolladores puedan integrar Twitter en sus servicios o aplicaciones. Según declaraciones de Twitter más del 50 % del tráfico de Twitter proviene de su API, lo que da una idea de la importancia de esta.

Twitter se define a sí mismo como una red de información y mecanismo de computación que produce más de 200 millones de Tweets o mensajes cortos al día. Se ofrece acceso a estos datos a través de la API proporcionada. Twitter ofrece 4 tipos de API:

- **Twitter para sitios web (TfW)**: se trata de un conjunto de productos que habilitan a las webs la integración de Twitter. Es ideal para integrar las funcionalidades más básicas de Twitter de manera rápida y sencilla. Estas funcionalidades abarcan incluir un botón de Twitter para publicar en la red social algo o un botón para seguir a una cuenta.
- **API de búsqueda (Search API)**: esta API está diseñada para productos que quieran hacer "querys" o búsquedas del contenido de Twitter. Esto puede ser encontrar un conjunto de Tweets por una palabra concreta o encontrar Tweets de un usuario.
- **REST API**: esta API es la que permite a los desarrolladores acceder a las funciones más básicas de Twitter como recuperar un Timeline, actualizar un estado o la información de usuario. Esta es la API usada en este proyecto ya que tratamos de imitar el funcionamiento general de Twitter en una aplicación propia. Todo se realiza con llamadas RESTful.
- **Streaming API**: Esta API está orientada a aquellos desarrolladores que necesitan hacer un uso intensivo de datos de Twitter. Es el caso de una aplicación que saque estadísticas de Tweets. La Streaming API permite mantener una conexión HTTP larga.

La API usada para el proyecto, la REST API permite operar sobre el Timeline recuperando Tweets, Retweets y menciones, sobre los mensajes, los favoritos, las actualizaciones de estado y en general ofreciendo todas las funcionalidades que Twitter ofrece en sus aplicaciones.

Existen además APIs de terceros recomendadas por Twitter para varios lenguajes de programación entre los que se incluyen .NET, Go, Java, PHP, Python y otros lenguajes. Esto permite implementar más rápidamente un servicio de Twitter en distintos entornos y aplicaciones.

Para este proyecto se accede a la REST API a través de la API para Java recomendada por Twitter Developers [19]: Twitter4J, [20] actualizada recientemente. Esta API además cuenta con soporte específico para Google App Engine y Android lo que la convierte en la API ideal para este proyecto.

3.MARCO REGULADOR

Dado que esta aplicación almacena datos de usuario, puede verse afectada por la Ley orgánica de protección de datos. En esta sección se analiza esa Ley para comprobar si afecta al proyecto.

Dicha ley, la Ley orgánica 15/1999 de Protección de Datos de Carácter Personal vela por los datos personales, libertados y derechos fundamentales de las personas físicas. Esta se encarga de garantizar y proteger poniendo especial interés en el honor, intimidad y privacidad de la persona y su familia.

Esta recoge además las obligaciones a las que deben atenerse los responsables de los ficheros que contienen dichos datos personales. Los responsables de los tratamientos de dichos datos, sean organismos públicos o privados deben garantizar el derecho a la protección de datos personales. Esto afecta a cualquier profesional, organización o empresa que guarde datos con carácter personal. Los datos que se consideran de carácter personal son datos como nombre y apellidos, fecha de nacimiento, dirección postal, DNI, una fotografía, etc… [21] [22]

Los datos almacenados en esta aplicación no afectan a esta ley ya que almacenan datos relativos a la compresión en el caso del servidor, y Tweets de usuarios en el caso de la aplicación. Las credenciales del usuario, así como su contraseña no se conocen debido al método de autenticación OAuth obligatorio en Twitter, que impide al desarrollador conocer a esta información.

4. ALGORITMOS DE COMPRESIÓN

El principal propósito de este proyecto es comprimir los datos todo lo posible, pero tiene un peso aún mayor que esta tarea se produzca en el menor tiempo posible. Para esto encontramos multitud de algoritmos con diferentes características, tasas y tiempos de compresión, pero en el caso de este proyecto se debe buscar un **algoritmo de compresión ligero** donde la velocidad de compresión y descompresión tienen un peso importante debido a que su uso va orientado a cantidades de datos no muy grandes que deben ser recibidas sin mucha espera y que operará sobre un dispositivo con bajas capacidades de computación como es un Smartphone.

Antes de analizar los algoritmos es importante conocer las limitaciones debido a las tecnologías usadas en este proyecto como son Android, Java, Google App Engine y Twitter. En primer lugar y ya que Java es el lenguaje seleccionado para el desarrollo tanto de Android como de Google App Engine, los algoritmos usados deben de soportar dicha plataforma. Entre ellos encontramos numerosas opciones y algunas realmente interesantes como el algoritmo LZO, un algoritmo que presume de conseguir las mismas tasas de compresión que el famoso ZIP pero con un tiempo mucho menor que este. Este algoritmo es el utilizado por ejemplo en el Curiosity enviado a Marte. [21]

Sin embargo algoritmos como este son descartados entre las opciones por no tener una **implementación pura en Java**, ya que este hace uso de algunas cabeceras y librerías de C a través de Java y Google App Engine sólo permite la ejecución de Java puro.

Por tanto se analizan algunos de los mejores algoritmos de compresión ligeros escritos en Java. Para el análisis se comprimen por igual los siguientes datos:

- Un **array de ceros** de un tamaño aproximado a las cadenas que posteriormente son utilizadas. Aquí se buscan las mayores tasas de compresión de cada algoritmo.
- Un array de tamaño igual al anterior con **caracteres generados aleatoriamente**. Aquí se busca el comportamiento de cada algoritmo en la peor situación posible.
- Un **texto** escogido aleatoriamente con palabras del castellano, almacenado en un String.
- Un conjunto de **20 Tweets** recogidos directamente desde la API y almacenados en tipo de dato List() de Java.
- El mismo conjunto de **20 Tweets** usado anteriormente pero **serializados**, almacenados en un array de bytes. Este será el caso más relevante por ser el usado en el proyecto.

Todos estos datos son comprimidos y descomprimidos por igual por los siguientes algoritmos:

- **LZF en bloque**: LibLZF es muy ligero y pensado donde el nivel de compresión es importante pero la velocidad de compresión lo es aún más. Su uso es tremendamente simple tratando de reducirse a comprimir y descomprimir. Está orientado a ser usado para compresión en tiempo real donde puede ahorrar algunos ciclos de CPU.

- **LZF en stream**: El algoritmo es el mismo que el anterior salvo que en vez de comprimir en bloque, lo hace haciendo uso de un stream de bytes.
- **QuickLZ en niveles 1 y 3 de compresión**: Este algoritmo solo comprime en bloque y dice ser uno de los más rápidos. Se puede adquirir una licencia o usarlo con GPL. Los benchmarks en su web prometen ratios de compresión y velocidades sorprendentes respecto a otros algoritmos. En el caso de los niveles de compresión se da prioridad a la tasa de compresión o a la velocidad según el nivel.
- **JZlib**: Este algoritmo no es más que una reimplementación del algoritmo zlib pero en Java. El algoritmo es gratuito, libre y sin restricciones legales.
- **BZip2**: Este algoritmo está incluido en la librería para compresión de Apache Commons Compress que incluye dump, tar, zip, gzip, xz, pack200, UNIX dump, cpio, ar y bzip2. Este está licenciado bajo BSD y dice ser mejor que algoritmos como Gzip o WinZip pero también usa más memoria y tiempo para su ejecución.
- **GZip**: El algoritmo abreviatura de GNU ZIP reemplaza al programa compress de UNIX y está basado en el algoritmo Deflate. Este también está incluido en las librerías de Apache.

Los resultados obtenidos al probar los datos propuestos con los algoritmos mencionados son observados en las siguientes tablas. Para los cálculos se creó un programa en Java a modo de "Benchmark" y se hicieron 10 tomas de los datos, descartando los datos atípicos, para finalmente calcular la media.

4.1 Tasa de compresión

La tasa de compresión mide el % de bytes ahorrados respecto al número de bytes original al comprimir.

Tabla 3 Tasa de compresión según algoritmos

TASA DE COMP.	LZF Block	LZF Stream	QuickLZ 1	QuickLZ 3	JZlib	Bzip2	Gzip
Ceros	99 %	99 %	99 %	98 %	100 %	100 %	100 %
Random	0 %	0 %	0 %	0 %	21 %	22 %	21 %
Texto	37 %	37 %	35 %	42 %	55 %	57 %	55 %
Tweets	75 %	75 %	80 %	82 %	86 %	85 %	86 %
Tweets Serial.	61 %	61 %	66 %	69 %	74 %	71 %	74 %

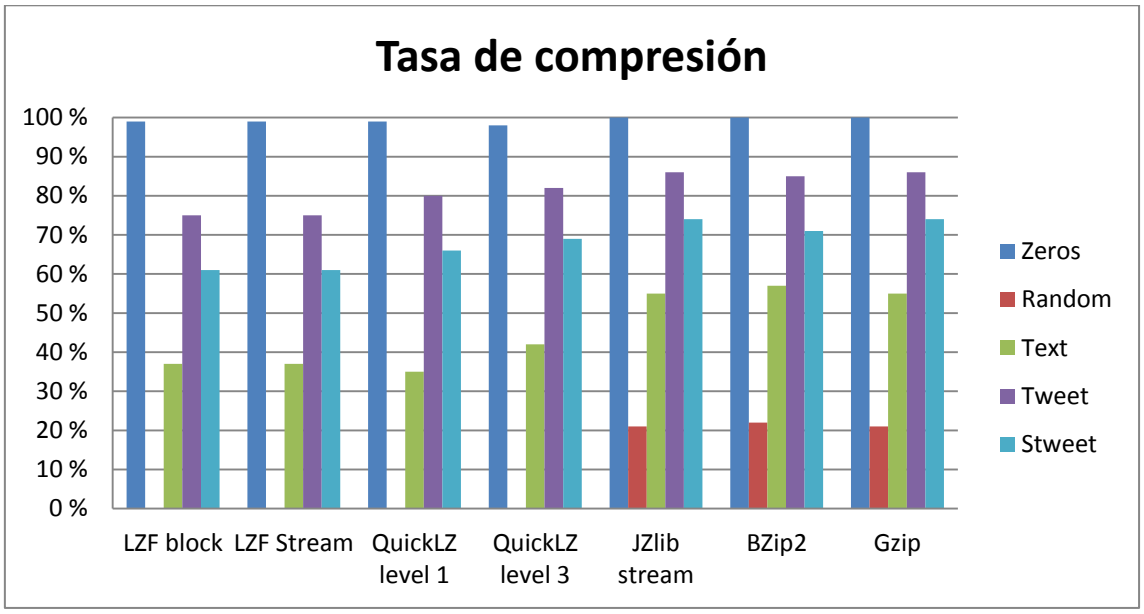

Ilustración 28 Tasa de Compresión

A la vista de los resultados de las gráficas se deben analizar con mayor detenimiento los datos de la columna **Stweet**, ya que son el tipo de datos que se usará en el proyecto. La diferencia con los Tweet es la serialización, donde esta es necesaria para transmitir los datos por la red. Por ello a pesar de que las tasas de compresión sean mayores para los datos de Tweet, los datos más relevantes son los de Stweet.

Los dos algoritmos con mejores tasas de compresión son el JZlib y Gzip, seguidos de cerca por el resto de algoritmos, especialmente por BZip2. En este caso el algoritmo LZF en sus dos variantes se queda alrededor del 60% de compresión mientras que los dos mejores se encuentran alrededor del 75%.

Esto es extrapolable al resto de pruebas, donde los ceros representarán la capacidad máxima de compresión del algoritmo y Random el peor caso con el que puede encontrarse.

3.2 Tiempo de compresión

El tiempo que tarda en comprimir se mide en **nanosegundos**. Esto se debe a que los tiempos medidos en milisegundos en multitud de casos eran 0, siendo imposible así comparar datos en los que los tiempos eran 0 para varios casos. Se han adaptado a notación científica por la longitud de las cifras para que puedan se puedan entender de manera más clara.

Tabla 4 Tiempo de compresión

TIEMPO COMP.	LZF Block	LZF Stream	QuickLZ 1	QuickLZ 3	JZlib	Bzip2	Gzip
Ceros	6.87e6	1.25e6	5.81e6	8.85e6	3.17e7	3.30e7	2.22e6
Random	1.81e7	1.53e7	1.57e7	3.88e7	1.20e8	2.24e8	7.46e6
Texto	1.28e6	8.42e5	1.56e6	3.47e6	2.95e6	9.16e6	4.93e5
Tweets	9.42e5	1.11e6	7.64e6	1.92e7	4.53e6	1.90e7	1.51e6
Tweets Serial.	9.42e5	2.66e5	3.49e6	8.38e6	4.42e6	1.45e7	1.23e6

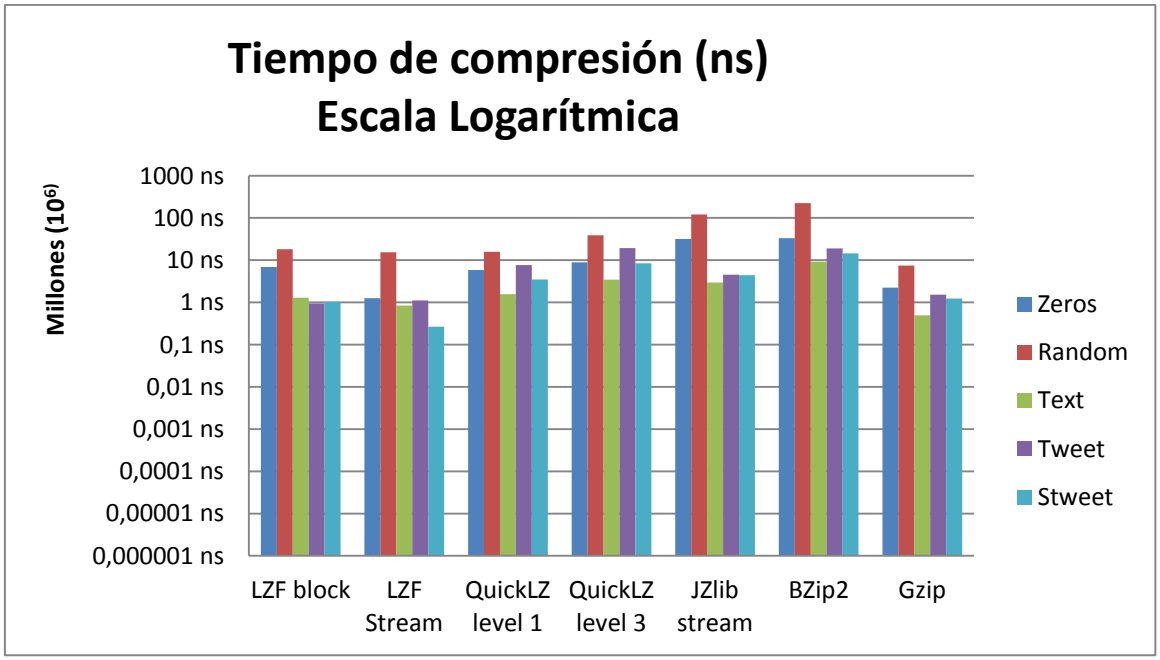

Ilustración 29 Tiempo de compresión Logarítmica

Analizar la gráfica resultante de estos datos es complicado debido al desfase entre los datos de los Tweets y los datos del array aleatorio, que tarda muchísimo más que el resto de casos. Por

ello se muestra en escala logarítmica donde las pequeñas distancias observables pueden representar millones de nanosegundos y se ofrece una segunda gráfica eliminando los datos del caso aleatorio y el caso de los ceros, para analizar los casos de datos reales entre los algoritmos.

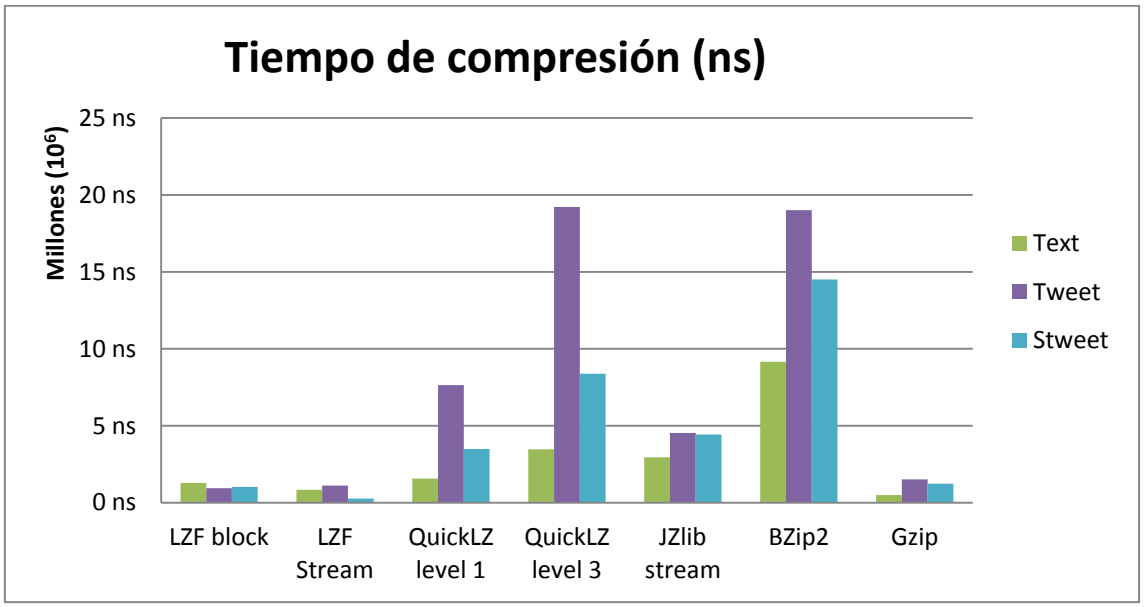

Ilustración 30 Tiempo de compresión

Con estos datos podemos ver como los mejores casos en cuanto a velocidad de compresión son para el algoritmo LZF en sus dos variantes y Gzip, con diferencias mínimas entre ambos. Para el caso de los Stweet, el algoritmo más rápido es el LZF en Stream.

El resto de algoritmos consiguen tiempos varias veces más lentos. Sin olvidar que los datos son analizados en nanosegundos y por tanto las diferencias entre los algoritmos son mínimas, al realizar estas operaciones en el servidor multitud de veces, para multitud de peticiones, repercutirá finalmente en la carga de trabajo del servidor el ahorro de nanosegundos.

3.3 Tiempo de descompresión

El tiempo que tarda en descomprimir se ha calculado de la misma forma que al comprimir.

Tabla 5 Tiempo descompresión

TIEMPO DESCOMP.	LZF Block	LZF Stream	QuickLZ 1	QuickLZ 3	JZlib	Bzip2	Gzip
Ceros	6.48e6	1.28e6	2.84e6	3.06e6	1.56e7	2.65e7	2.84e6
Random	8.42e4	1.23e5	5.75e4	8.55e4	2.35e7	4.10e7	1.24e6
Texto	7.70e5	5.61e5	2.31e6	1.13e5	1.86e6	4.18e6	1.50e5
Tweets	2.22e6	1.88e6	4.42e6	2.23e5	3.95e6	4.14e6	5.32e5
Tweets Serial.	1.99e6	1.64e6	3.58e6	1.59e5	4.18e6	2.72e6	3.95e5

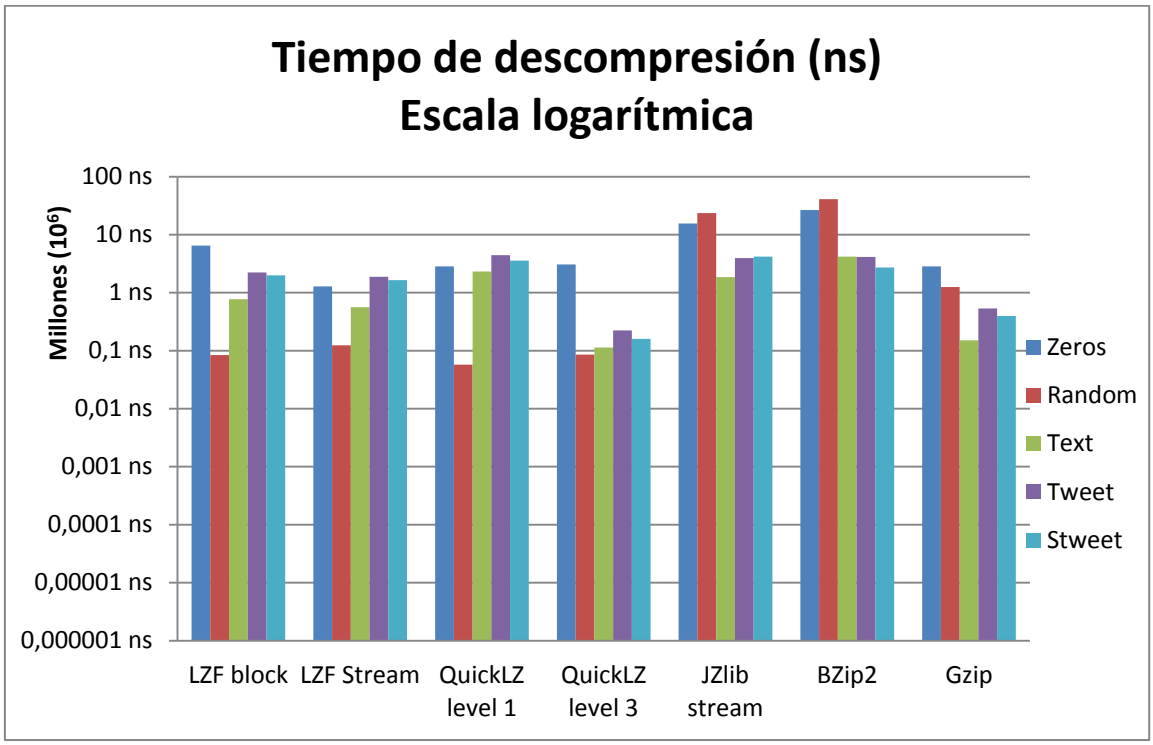

Ilustración 31 Tiempo de descompresión Logarítimca

Al igual que el caso de la compresión de datos, los datos aleatorios y el array de ceros hace que la escala deba mostrarse logarítmicamente y además se muestra una en la que se eliminan el Random y los Zeros:

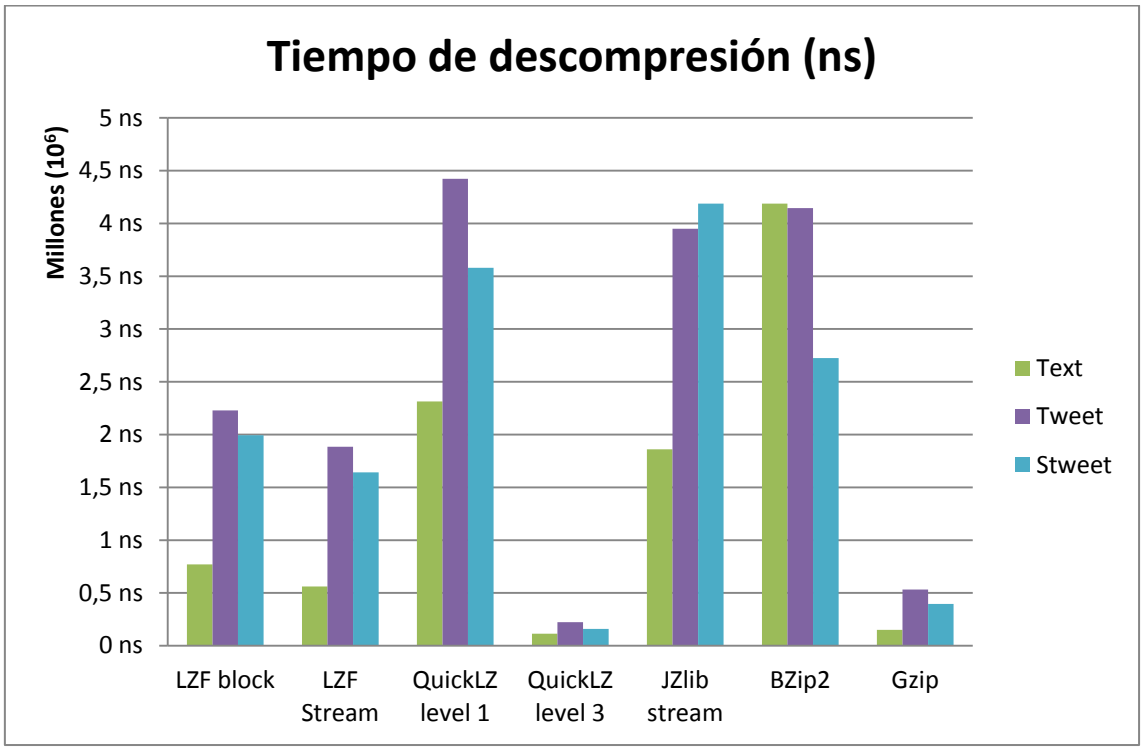

Ilustración 32 Tiempo de descompresión

Con estos datos si podemos observar claramente como los algoritmos más rápidos a la hora de descomprimir son el QuickLZ en nivel 3 y el Gzip, con el resto de algoritmos varias veces por encima en cuanto a tiempos.

Los tiempos de descompresión repercuten en la aplicación móvil, la cual tiene una capacidad de cómputo pequeña, pero además se trata de un dispositivo con batería, donde un menor uso de recursos se traduce en una mayor duración de la batería, por tanto elegir el algoritmo con menor tiempo de descompresión es importante para el dispositivo móvil.

3.4 Conclusiones sobre los algoritmos

A la vista de los resultados obtenidos se debe escoger el algoritmo más equilibrando, dando importancia a las velocidades de compresión y descompresión sin olvidar la tasa de compresión de cada algoritmo. Como ya se ha visto cuando se trata de comprimir los datos los dos mejores algoritmos son JZlib y Gzip. Cuando se trata de conseguir la mejor velocidad al comprimir son el LZF y el Gzip. Y a la hora de conseguir la mejor velocidad descomprimiendo, son el QuickLZ y el Gzip los que destacan claramente del resto.

Sin duda el algoritmo Gzip, sin ser el mejor en todas las pruebas se trata del algoritmo con mejor compromiso en cuanto a los tres factores analizados, y por tanto es el algoritmo elegido para este proyecto para conseguir buenas tasas de compresión a las mejores velocidades posibles.

5. ANÁLISIS

En esta sección se expone un análisis con mayor detalle del sistema para que sea más sencillo crear los posteriores diseños y el desarrollo del sistema.

5.1 Detalle de la aplicación

El objetivo de la aplicación es permitir un funcionamiento básico de Twitter en un Smartphone con Android, con una interfaz agradable e intuitiva para el usuario. El funcionamiento básico incluirá lo siguiente:

- La posibilidad de autenticarse de manera segura con un usuario y contraseña registrados en Twitter.
- Actualizar el estado de Twitter, esto es, la posibilidad de Twittear un mensaje corto de 140 caracteres.
- Actualizar el Timeline del usuario autenticado, lo que recogerá en base a la API los 20 últimos Tweets de los usuarios a los que se sigue, y poder visualizarlos de manera intuitiva.
- Iniciar un servicio que actualice de manera automática el Timeline cada unos determinados minutos.

Por otra parte este proyecto trata de añadir funcionalidades ahorrando los datos consumidos por el Smartphone al hacer uso de las funcionalidades básicas. Para ellos será necesario un servidor o Proxy totalmente transparente al usuario que se encargará de la compresión de los datos. Las funcionalidades extra deben cumplir lo siguiente:

- Compresión del Timeline para reducir el número de bytes que se transmiten por la red.
- Configuración de la aplicación para reducir más o menos el consumo de datos en función de la selección del usuario.
- Visualización gráficamente de la cantidad de datos ahorrados al usar el Proxy.

5.2 Requisitos software

A continuación se exponen los requisitos que detallan las características indispensables que debe tener el sistema que propicien el buen funcionamiento del sistema.

La tabla explicativa se compone de diversos campos:

- **ID Requisito**: identificador unívoco del requisito.
- **Tipo**: funcional o de usabilidad, según corresponda de acuerdo a su cometido.

- **Evento/Caso de uso**: situación que debe ocurrir como disparador.
- **Descripción**: breve explicación del significado del requisito.
- **Justificación**: por qué es importante el requisito y qué motivación lo ha propiciado.
- **Criterio de cumplimiento**: comprobante de una implementación del requisito satisfactoria.
- **Prioridad**: relevancia del requisito.
- **Conflictos**: requisitos que no pueden ser implementados si ése requisito es implementado.

Los requisitos son los siguientes:

Tabla 6 RF-01

ID Requisito	RF-01	**Tipo**	Funcional
Evento / Caso de uso	Autenticarse en la aplicación.		
Descripción	El usuario introduce su usuario y contraseña en la web de Twitter para dar las credenciales de OAuth a la aplicación.		
Justificación	Las credenciales proporcionadas son necesarias para cualquier petición del usuario a la API de Twitter.		
Criterio de cumplimiento	Lanzar un navegador, que el usuario introduzca sus datos y al volver capturar las credenciales.		
Prioridad	Alta	**Conflictos**	#

Tabla 7 RU-01

ID Requisito	RU-01	**Tipo**	Usabilidad
Evento / Caso de uso	Selección entre varios niveles de compresión.		
Descripción	El usuario a través del menú de preferencias selecciona entre 4 niveles de compresión.		
Justificación	El usuario en función de las características de su móvil, su conexión y su cobertura puede seleccionar un nivel de compresión.		
Criterio de cumplimiento	Abrir el menú de opciones y seleccionar la opción preferencias, y de entre ellas seleccionar una.		
Prioridad	Media	**Conflictos**	#

Tabla 8 RF-02

ID Requisito	RF-02	Tipo	Funcionalidad
Evento / Caso de uso	Selección del nivel de compresión: No compression.		
Descripción	Al seleccionar la opción No compression, se desactiva el Proxy haciendo todas las peticiones de manera directa a Twitter. Además las imágenes de usuario se descargan en su máxima calidad.		
Justificación	Si el usuario dispone de conexión Wi-Fi o buena cobertura puede seleccionar esta opción.		
Criterio de cumplimiento	Una vez seleccionada la opción, al actualizar el Timeline se muestran los avatares en alta resuloución y los Tweets se piden directamente a Twitter.		
Prioridad	Alta	Conflictos	#

Tabla 9 RF-03

ID Requisito	RF-03	Tipo	Funcionalidad
Evento / Caso de uso	Selección del nivel de compresión: Compression + big avatar.		
Descripción	Al seleccionar la opción Compression + big avatar, el Proxy se activa, se hace una petición al servidor, este hace dicha petición a Twitter, comprime lo recibido y el cliente descarga los datos comprimidos. Los avatares se muestran al usuario en alta resolución.		
Justificación	El usuario puede querer comprimir los Tweets pero ver los avatares en alta calidad.		
Criterio de cumplimiento	Una vez seleccionada la opción, al actualizar el Timeline se muestran los avatares en alta resuloución y los Tweets llegan comprimidos a través del servidor.		
Prioridad	Alta	Conflictos	#

Tabla 10 RF-04

ID Requisito	RF-04	Tipo	Funcionalidad
Evento / Caso de uso	Selección del nivel de compresión: Compression + little avatar.		
Descripción	Al seleccionar la opción Compression + little avatar, el Proxy se activa, se hace una petición al servidor, este hace dicha petición a Twitter, comprime lo recibido y el cliente descarga los datos comprimidos. Los avatares se muestran al usuario en baja resolución, descargando así menos datos.		
Justificación	El usuario puede querer comprimir los Tweets y ver los avatares de cada usuario, pero sacrifica la calidad de la imagen para ahorrar datos.		
Criterio de cumplimiento	Una vez seleccionada la opción, al actualizar el Timeline se muestran los avatares en baja resuloución y los Tweets llegan comprimidos a través del servidor.		
Prioridad	Alta	Conflictos	#

Tabla 11 RF-05

ID Requisito	RF-05	Tipo	Funcionalidad
Evento / Caso de uso	Selección del nivel de compresión: Max Compression.		
Descripción	Al seleccionar la opción Max Compression, el Proxy se activa, se hace una petición al servidor, este hace dicha petición a Twitter, comprime lo recibido y el cliente descarga los datos comprimidos. Los avatares no se muestran al usuario, evitando así la descarga de datos asociados a cada avatar.		
Justificación	Para ahorrar el máximo número posible de datos sin renunciar a las funcionalidades de Twitter, o en situaciones de mala cobertura, el usuario debe poder comprimir los Tweets y evitar la descarga de imágenes para optimizar al máximo sus descarga de datos.		
Criterio de cumplimiento	Una vez seleccionada la opción, al actualizar el Timeline se muestran los avatares en baja resolución y los Tweets llegan comprimidos a través del servidor.		
Prioridad	Alta	Conflictos	#

Tabla 12 RF-06

ID Requisito	RF-06	Tipo	Funcionalidad
Evento / Caso de uso	Actualización el Timeline automáticamente sin interacción del usuario.		
Descripción	Se debe poder actualizar el Timeline del usuario cada cierto tiempo de manera que cuando el usuario accede a la aplicación esta está actualizada en un rango de tiempo.		
Justificación	Para que el usuario no tenga que hacer una actualización de un gran número de Tweets si ha pasado demasiado tiempo, debe tener la opción de iniciar un servicio en segundo plano que actualice automáticamente y cada cierto tiempo el Timeline.		
Criterio de cumplimiento	Cada 10 minutos el Timeline se actualiza sin que el usuario interaccione si el servicio está iniciado.		
Prioridad	Baja	Conflictos	#

Tabla 13 RU-02

ID Requisito	RU-02	Tipo	Usabilidad
Evento / Caso de uso	Actualización el Timeline manualmente.		
Descripción	Se debe poder actualizar el Timeline del usuario de forma manual y en cualquier momento mediante un botón en la aplicación.		
Justificación	El usuario debe poder consultar los últimos Tweets del Timeline en cualquier momento por lo que debe poder actualizar cuando quiera.		
Criterio de cumplimiento	Al pulsar el botón Update Timeline en el menú de opciones, el Timeline se actualiza con los últimos Tweets.		
Prioridad	Alta	Conflictos	#

Tabla 14 RU-03

ID Requisito	RU-03	Tipo	Usabilidad
Evento / Caso de uso	Descargar Tweets comprimidos del Proxy.		
Descripción	Cuando una de las opciones de compresión está activada, al actualizar el Timeline se pide al Proxy que los descargue y se descargan entonces del Proxy.		
Justificación	Esta es la opción que permite que los Tweets se descarguen comprimidos, tienen que pasar previamente por el Proxy.		
Criterio de cumplimiento	Si una opción de compresión está activada al actualizar la conexión se establece con el Proxy y no con Twitter de manera directa.		
Prioridad	Alta	Conflictos	#

Tabla 15 RU-04

ID Requisito	RU-04	Tipo	Usabilidad
Evento / Caso de uso	Desplazar el Timeline con un gesto.		
Descripción	La lista de Tweets se debe desplazar con un gesto del dedo para poder visualizarla completamente.		
Justificación	Cuando se actualiza el Timeline este tendrá al menos 20 Tweets, cuya visualización simultánea es imposible por lo que se establece un mecanismo para desplazarse por la lista de Tweets.		
Criterio de cumplimiento	En un Timeline actualizado al deslizar el dedo sobre la pantalla horizontalmente se desplaza la lista mostrando Tweets anteriores o posteriores según el gesto.		
Prioridad	Alta	Conflictos	#

Tabla 16 RF-07

ID Requisito	RF-07	Tipo	Funcionalidad
Evento / Caso de uso	Actualización del estado de la cuenta del usuario.		
Descripción	En cualquier momento el usuario debe poder actualizar su estado en forma de mensaje corto de una longitud máxima de 140 caracteres.		
Justificación	Una de las funciones básicas de Twitter es publicar un Tweet y es lo que permite que la red social tenga utilidad, las publicaciones de los usuarios.		
Criterio de cumplimiento	Al pulsar el botón Update Status en el menú de opciones de la aplicación debe accederse a una pantalla que permita escribir el Tweet y un botón que lo publique.		
Prioridad	Alta	Conflictos	#

Tabla 17 RU-05

ID Requisito	RU-05	Tipo	Usabilidad
Evento / Caso de uso	Un contador disminuye según se escriben caracteres para actualizar un estado.		
Descripción	Cuando se escribe un Tweet para ser publicado se informa al usuario del número de caracteres restantes. El color del número cambiará a amarillo al quedar pocos caracteres y a rojo cuando se ha llegado al límite de 140 caracteres.		
Justificación	Ya que el límite del mensaje corto son 140 caracteres el usuario debería conocer visualmente el espacio restante para la publicación.		
Criterio de cumplimiento	Al escribir varios caracteres el contador disminuye en función de los caracteres escritos y cambiar de color según la descripción.		
Prioridad	Baja	Conflictos	#

Tabla 18 RU-06

ID Requisito	RU-06	**Tipo**	Usabilidad
Evento / Caso de uso	Visualización de estadísticas con información sobre los bytes consumidos con y sin compresión.		
Descripción	El usuario debe poder consultar unas estadísticas sobre cuantos bytes está ahorrando al hacer uso del proxy y que cantidad de bytes consumiría si no hiciese uso del mismo para comparar y visualizar gráficamente el beneficio de la aplicación.		
Justificación	Con el fin de comprobar que la aplicación funciona y el ahorro es significativo para el usuario, debe poder consultar los datos relativos a los bytes utilizados.		
Criterio de cumplimiento	Al pulsar el botón Statistics en el menú de opciones se llega a una pantalla que muestra la gráfica correspondiente.		
Prioridad	Alta	**Conflictos**	#

Tabla 19 RU-07

ID Requisito	RU-07	**Tipo**	Usabilidad
Evento / Caso de uso	Visualización de estadísticas con información sobre la tasa de compresión en cada actualización de Timeline.		
Descripción	El usuario debe poder consultar unas estadísticas sobre la tasa de compresión conseguida al actualizar el Timeline. Estas estadísticas informan al usuario de en qué casos se consiguen mejores tasas de compresión.		
Justificación	El usuario debe conocer en qué condiciones el Proxy tiene un funcionamiento óptimo para poder cambiar sus hábitos a la hora de actualizar el Timeline con el fin de optimizar los bytes descargados.		
Criterio de cumplimiento	Al pulsar el botón Statistics en el menú de opciones se llega a una pantalla que muestra la gráfica correspondiente.		
Prioridad	Media	**Conflictos**	#

Tabla 20 RF-08

ID Requisito	RF-08	Tipo	Funcionalidad
Evento / Caso de uso	Al actualizar el Timeline, si una imagen fue previamente descargada, se reutiliza a modo de caché, a pesar de que se cierre la aplicación.		
Descripción	Cada imagen descargada como avatar para cada Tweet se almacena de forma persistente. Cada imagen a utilizar se comprueba si fue desacargada anteriormente buscando en la lista de imágenes descargadas y si existe se carga de la caché, ahorrando los datos utilizados por la aplicación.		
Justificación	Para ahorrar aún más datos, las imágenes han de ser cacheadas, de forma que las imágenes que se usan habitualmente solo se descargan de internet una vez.		
Criterio de cumplimiento	Cuando se actualiza la interfaz con Tweets de usuarios que se han visualizado previamente la imagen no se descarga de Twitter.		
Prioridad	Media	Conflictos	#

Tabla 21 RF-09

ID Requisito	RF-09	Tipo	Funcionalidad
Evento / Caso de uso	Al actualizar el Timeline, las imágenes más utilizadas se cargan directamente desde memoria si se han cargado en esta antes.		
Descripción	A modo de caché temporal las imágenes más utilizadas se cargan en memoria RAM para que si vuelve a aparecer se cargue de memoria, siendo mucho más rápido y menos costoso que cargarlo de almacenamiento interno.		
Justificación	Ya que muchas imágenes son reutilizadas, si estás son almacenadas en RAM se ahorrará batería al optimizar el proceso de carga. Además la aplicación funcionará de forma más fluida.		
Criterio de cumplimiento	Cuando se actualiza la interfaz con Tweets de usuarios que se han visualizado previamente la imagen se carga de la memoria RAM si está disponible, en caso contrario se busca en almacenamiento interno.		
Prioridad	Baja	Conflictos	#

Tabla 22 RF-10

ID Requisito	RF-10	Tipo	Funcionalidad
Evento / Caso de uso	El servidor comprime los Tweets descargados.		
Descripción	Cuando el servidor recibe una petición para descargar Tweets, este hace la petición a Twitter, los descarga, los comprime y los manda al cliente.		
Justificación	Para ahorrar datos es necesario que el servidor los comprima antes de ser mandados al cliente.		
Criterio de cumplimiento	Cuando el cliente hace una petición al servidor este automáticamente hace la petición a Twitter, descarga la información solicitada y la comprime. El cliente recibe los Tweets comprimidos		
Prioridad	Alta	Conflictos	#

Tabla 23 RF-11

ID Requisito	RF-11	Tipo	Funcionalidad
Evento / Caso de uso	El servidor almacena los datos relativos a las peticiones del cliente.		
Descripción	Cada vez que el cliente hace una petición el servidor almacena la fecha y hora de la petición, el número de bytes sin comprimir y comprimidos y el identificador del usuario que hace la petición.		
Justificación	Para poder generar las estadísticas es necesario almacenar los datos de estas. Ya que el dispositivo móvil posee capacidades de almacenamiento limitadas esta información se almacena en el servidor.		
Criterio de cumplimiento	En cada petición del cliente el servidor hace persistentes los datos en una BBDD.		
Prioridad	Alta	Conflictos	#

Tabla 24 RF-12

ID Requisito	RF-12	Tipo	Funcionalidad
Evento / Caso de uso	El servido procesa los datos de las estadísticas antes de mandarlos al cliente.		
Descripción	El servidor calcula en base a los datos almacenados las tasas de compresión, la cantidad total de datos ahorrados y la diferencia de uso para mandarlas al cliente.		
Justificación	Ya que las estadísticas las descargará en cualquier caso el cliente, la computación asociada a las estadísticas se realiza en el servidor para descargar al cliente y ahorrar batería.		
Criterio de cumplimiento	Cuando se solicitan las estadísticas estas llegan procesadas del servidor.		
Prioridad	Media	Conflictos	#

Tabla 25 RF-13

ID Requisito	RF-13	**Tipo**	Funcionalidad
Evento / Caso de uso	La aplicación pide un número concreto de Tweets.		
Descripción	La aplicación solicita los Tweets disponibles desde el último Tweet que recibió.		
Justificación	Con esto se consigue ahorrar datos, evitando la descarga innecesaria de Tweets descargados previamente.		
Criterio de cumplimiento	Al realizar una petición no se descargan Tweets existentes.		
Prioridad	Media	**Conflictos**	#

6. DISEÑO

En esta sección se explica el diseño del sistema a desarrollar con el objetivo de definir la arquitectura y los componentes del sistema.

6.1 Lenguaje de programación

Para el diseño del sistema se han tenido en cuenta varios lenguajes de programación para cada uno de los subsistemas de los que está compuesto, el servidor y la aplicación móvil. Cada subsistema debe desarrollarse en el lenguaje de programación que mejor se adapte al sistema.

En la elección de dichos lenguajes se ha tenido en cuenta la compatibilidad con distintas bibliotecas y especialmente la compatibilidad entre los dos subsistemas.

6.1.1 Aplicación móvil

El lenguaje de programación elegido para la aplicación móvil quedaba bastante restringido al seleccionar Android como sistema operativo, ya que este se desarrolla casi en su totalidad en Java y el SDK ofrecido por Google se sirve de este lenguaje. Sin embargo existe la posibilidad de utilizar el lenguaje C para partes de la aplicación bien a través de Java Native Interface (JNI) o bien a través del Native Development Kit (NDK) proporcionado por Google.

El uso de estas herramientas permitiría incluir las bibliotecas de algunos algoritmos de compresión que sólo se encuentran disponibles en código C.

Sin embargo, por limitaciones en el servidor el lenguaje de programación usado en Android será **Java**.

Java se trata de un lenguaje de alto nivel orientado a objetos. El código Java en Android tiene la particularidad de no ejecutarse de la manera habitual ya que en Android no existe una JVM como tal. En Android el código se compila en un ejecutable Dalvik, que se trata de una máquina virtual especializada para Android y optimizada para dispositivos móviles.

6.1.2 Servidor

El servidor web actuará de manera totalmente transparente al usuario, sin que este tenga interacción con el mismo. Por esto no es necesario diseñar una interfaz para el mismo y esto reduce los posibles lenguajes de programación como el uso de HTML y CSS para la interfaz.

Al elegir **Google App Engine** por su sencilla escalabilidad y fiabilidad en cuanto al tiempo que está disponible, los lenguajes quedan limitados a Java, Python y Go. Por compatibilidad entre ambos sistemas en cuanto a las bibliotecas usadas para los algoritmos de compresión y API de Twitter el lenguaje de programación elegido será Java.

6.2 Base de datos

Cada uno de los subsistemas de este proyecto necesita de una base de datos para hacer persistentes varios datos. En este apartado se diseñarán las estructuras de las bases de datos.

6.2.1 Aplicación móvil

El sistema operativo Android incorpora por defecto soporte para la base de datos SQLite, la cual se comporta de manera adecuada para dispositivos móviles que por lo general no tienen grandes capacidades de almacenamiento. **SQLite** no requiere de instalación ni configuración ya que viene totalmente integrado en Android y además solo utiliza un fichero multiplataforma para almacenar los datos. SQLite es muy compacta y almacena los datos en registros de longitud variable para que los datos ocupen su tamaño real.

En la aplicación móvil se necesitan almacenar dos conjuntos de datos:

Por un lado se almacenan datos de autenticación de Twitter, que son varios Tokens, y además se almacenan las opciones seleccionadas en las preferencias. Para esto se utiliza la utilidad de Android “SharedPreferences”, que no cuenta con tablas y se usa para almacenar pequeños datos.

Por otro lado deben almacenarse los Tweets, para los cuales si será necesaria una BBDD que proporcionará los datos a la aplicación por medio de un Content Manager. El diseño de dicha BBDD es el de la Tabla 26 Timeline BBDD.

Tabla 26 Timeline BBDD

Timeline	
PK	**_id**
	created_at txt user profile_url

Como se puede observar el diseño no puede ser más simple. Está compuesto por una sola tabla que almacena los datos necesarios asociados a los Tweets. El significado de los campos es el siguiente:

- **_id**: la clave primaria de la tabla será un identificador único para cada Tweet. Este identificador viene dado por la API de Twitter y no puede repetirse a no ser que se almacene el mismo Tweet, lo cual se evita comprobando antes si existe.
- **Created_at**: se trata del momento en milisegundos desde 1970 de creación del Tweet, que servirá para ordenarlos cronológicamente.
- **Txt**: el contenido del Tweet que se mostrará en la interfaz.
- **User**: el nombre de usuario asociado al Tweet que permite identificar el autor del Tweet y que se mostrará también en la interfaz. Será indispensable en la interfaz que no muestra los avatares.
- **Profile_url**: es la URL que apunta al avatar asociado al usuario y que servirá para descargar dicha imagen cuando la interfaz lo solicite. Aquí se almacenará una URL distinta en función de las preferencias de usuario, una de alta calidad, baja calidad o la original.

Las operaciones básicas que se realizan con esta BBDD utilizando una interfaz que simplificará la sintaxis de SQL en una clase llamada DbHelper. Esta clase seguirá el sistema CRUD utilizando tan solo las cuatro operaciones básicas de una BBDD: Create, Read, Update y Delete. Estas operaciones son las siguientes:

- **Insert()**: permite insertar una o más filas en la base de datos. Es el comando asociado a Create.
- **Query()**: solicita una fila que concuerde con los criterios de búsqueda especificados. Es el comando asociado a Read.
- **Update()**: sustituye una o más filas que concuerden con los criterios especificados.
- **Delete()**: elimina las columnas que concuerden con el criterio especificado.

La razón para no usar SQL de forma directa son varias. En primer lugar por una razón de seguridad, una declaración en SQL es sensible de recibir ataques bien conocidos en nuestra aplicación, tales como los ataques de SQL injection. Además desde el punto de vista del rendimiento ejecutar declaraciones SQL repetidamente es muy ineficiente porque se tienen que parsear las SQL cada vez que se ejecuta el comando. Por último, utilizar esta capa de abstracción simplifica el uso y hace que se generen menos errores al compilar, y si existen sean más fáciles de identificar al estar todos centralizados en el mismo punto.

6.2.2 Servidor

El servidor, al ser Google App Engine utilizará los **Datastores** de Google que permiten almacenar grandes cantidades de datos que sean muy eficientes con aplicaciones escalables.

El App Engine Datastore carece de esquemas y tiene las siguientes características:

- Transacciones atómicas.
- NoSQL.
- Sin tiempo de inactividad planificado.

- Alta disponibilidad en lecturas y escrituras.
- Fuerte consistencia para lecturas y antiguas consultas.

A diferencia de las bases de datos relaciones tradicionales, el App Engine Datastore usa una arquitectura distribuida que automáticamente gestiona la escalabilidad de grandes cantidades de datos. Mientras que su interfaz ofrece muchas de las características de las bases de datos tradicionales, se diferencia de estas en la forma en la que describe las relaciones entre los objetos de datos.

App Engine Datastore está diseñado para escalar, permitiendo a las aplicaciones mantener un alto rendimiento aunque reciban mucho tráfico. Los tipos de búsqueda que se pueden realizar con los Datastores son más restrictivos, no pudiendo hacer operaciones Join, Inequality Filtering en múltiples propiedades y Filtering de datos basados en los resultados de otra query.

En la base de datos del servidor se almacenarán los datos de compresión en cada petición que se reciba, para calcular el ahorro de datos y cantidad de datos que se utiliza en cada petición.

Así pues se tendrá el siguiente diseño:

Tabla 27 Server BBDD

compData	
PK	**theHour**
	user uncompBytes compBytes

Donde los datos significan lo siguiente:

- **theHour**: será la clave primaria que almacenará el momento en el que se guarda dicha fila.
- **user**: es el identificador del usuario que solicita los Tweets.
- **uncompBytes**: la cantidad de bytes no comprimidos que se han utilizado en dicha petición.
- **compBytes**: la cantidad de bytes que mandan al cliente, los comprimidos.

6.3 Comunicaciones

La comunicación en el proyecto se realiza de manera distribuida entre los dos subsistemas, la aplicación móvil y el servidor. El servidor en este caso no se trata de un servidor al uso, con un solo PC corriendo la aplicación y gestionándose todo desde ahí. Al seleccionar un PaaS como se vio en secciones anteriores podríamos decir que se establece una comunicación entre la aplicación y la nube.

En este caso la nube es un sistema distribuido en el que pueden intervenir múltiples servidores, con la información y procesos muy repartidos, ya que esta es la característica esencial de los PaaS y lo que permite la escalabilidad.

El esquema sería el que se observa en la Ilustración 33 PaaS:

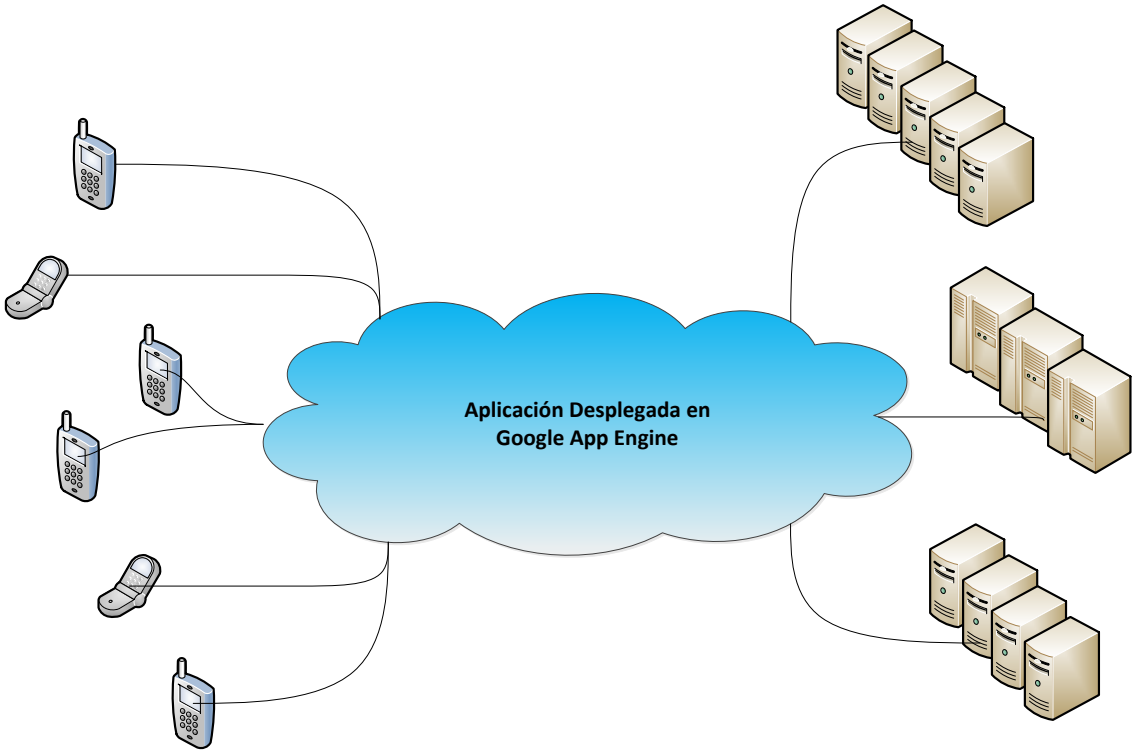

Ilustración 33 PaaS

En este caso los distintos dispositivos provistos de la aplicación se comunican con una nube que a su vez se compone de múltiples servidores distribuidos.

La comunicación se produce como consecuencia de una petición por parte de la aplicación, que puede enviar los siguientes mensajes:

- **Mensaje de solicitud de Tweets**: Este mensaje se compone de 4 Tokens que permitirán al servidor realizar la petición a Twitter y el identificador de un Tweet para solicitar los Tweets disponibles a partir de ese Tweet.
- **Mensaje de solicitud de estadísticas**: en este mensaje se recibe el ID del usuario que solicita las estadísticas y el servidor busca todos los datos asociados a este.

La aplicación esperará a recibir la confirmación de que el servidor recibió y procesó correctamente la petición y en cada caso pedirá la descarga de los Tweets comprimidos o las estadísticas.

En la Ilustración 34 Secuencia comunicaciones se puede apreciar el diagrama de secuencia para las peticiones.

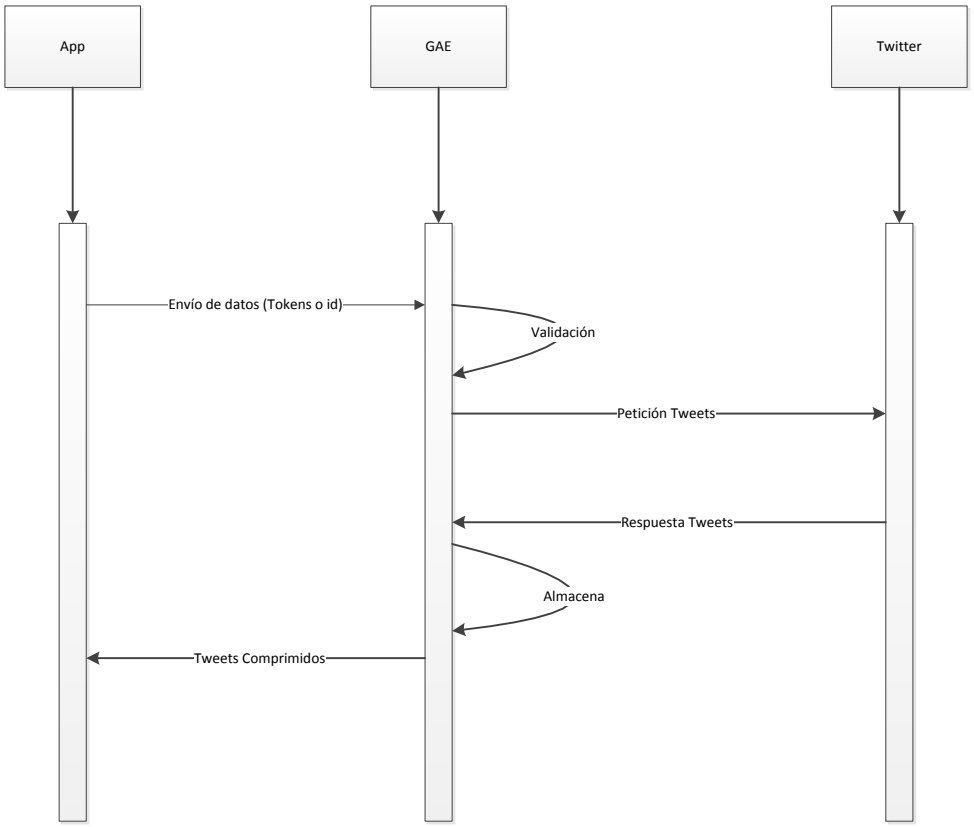

Ilustración 34 Secuencia comunicaciones

6.4 Aplicación Android

El diseño general de la aplicación puede observarse en la siguiente Ilustración 35 Diseño Aplicación Móvil:

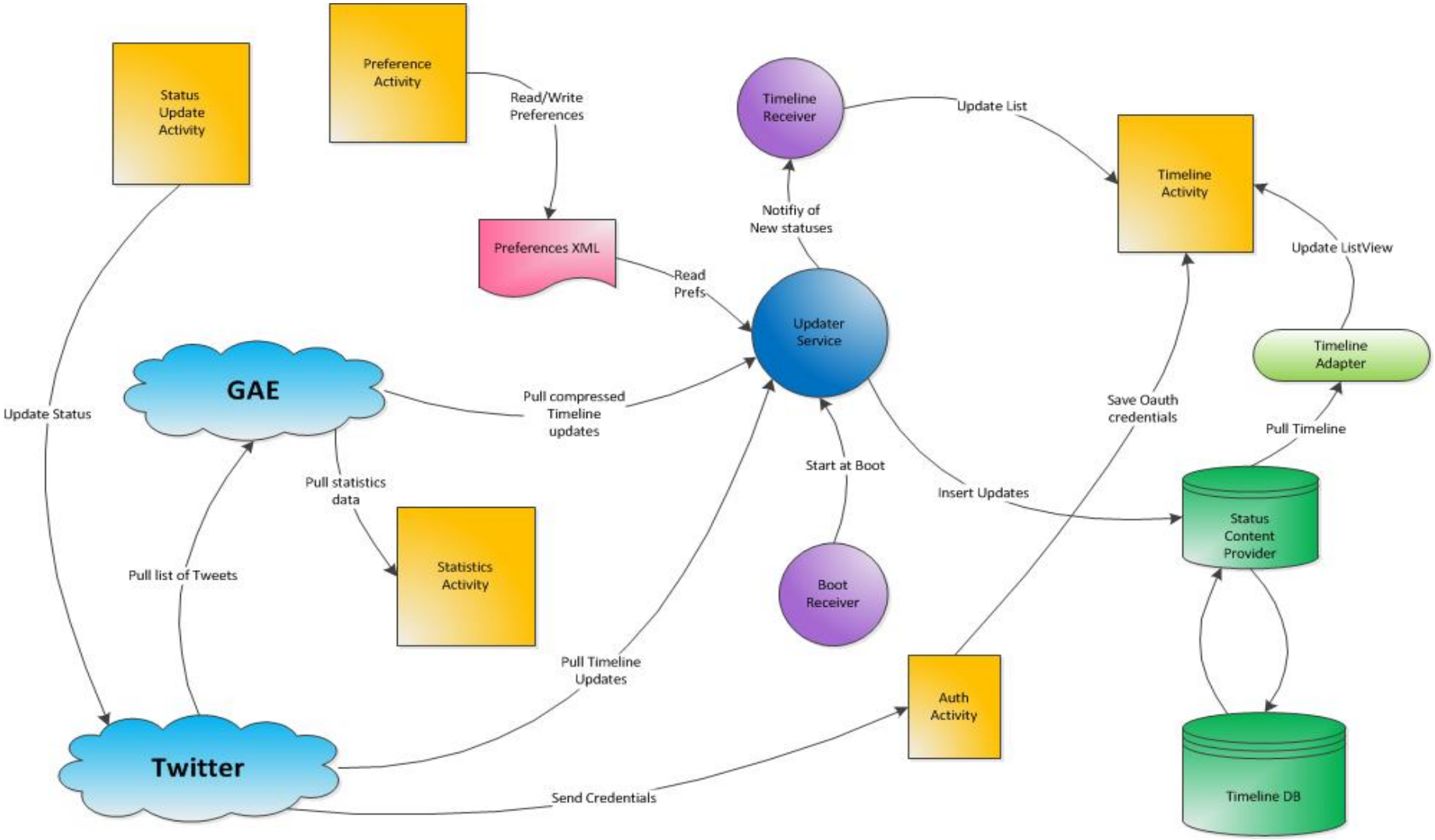

Ilustración 35 Diseño Aplicación Móvil

Aquí se explica cada componente de la aplicación:

- **Auth Activity**: es la "Main Activity" del proyecto, la primera en arrancar al iniciar la aplicación. Cuando este Activity no encuentra credenciales guardadas da la posibilidad de autenticarse en Twitter para guardarlas. En caso de encontrar credenciales guardadas pasa automáticamente a la Timeline Activity.
- **Timeline Activity**: la actividad principal donde ocurren la mayoría de las acciones. En este Activity se muestran los Tweets en una lista horizontal. Desde aquí se puede acceder al Activity que muestra las preferencias, al que muestra las estadísticas y al que permite actualizar el estado. Se puede actualizar el Timeline de forma manual y se puede iniciar y parar el servicio que actualiza el Timeline de forma automática.
- **Status Update Activity**: es el encargado de actualizar el estado del usuario o publicar un Tweet. Se comunica directamente con Twitter para mandar el estado.
- **Preference Activity**: permite seleccionar entre los distintos niveles de compresión en función de las necesidades del usuario. Estos cambios se guardan en el XML de preferencias.
- **Statistics Activity**: recibe las estadísticas del GAE y las muestra en forma de gráfica para visualizar el número de bytes consumidos y la tasa de compresión en cada petición.
- **Preferences XML**: es el archivo que muestra la opción de compresión seleccionada y permite elegir entre las opciones disponibles.
- **Updater Service**: es el servicio que se ejecuta en segundo plano para cada cierto tiempo actualizar los Tweets si hay nuevos disponibles. En función de la opción elegida en las preferencias recibe los Tweets directamente de Twitter sin comprimir o los recibe comprimidos del GAE. Cuando hay nuevos Tweets informa en broadcast de que existen nuevos Tweets y manda al gestor de Tweets (el Status Content Provider) los nuevos Tweets.
- **Timeline Receiver**: es el encargado de notificar al Timeline Activity de que existen nuevos Tweets para que actualice la interfaz con los nuevos Tweets. En realidad informa en broadcast a todos los Activities pero el único que lo procesa es el Timeline Activity.
- **Boot Receiver**: cuando el teléfono se inicia, este Receiver es el encargado de inciar el Updater Service automáticamente al arrancar el Smartphone.
- **Timeline Adapter**: es el componente que se encarga de adaptar los datos para el Timeline Activity de manera que lleguen correctamente. Por ejemplo para mostrar la hora correctamente o para mostrar una u otra interfaz en función de las preferencias.
- **Status Content Provider**: gestiona los datos de manera eficiente para ponerlos a disposición de cualquier otro componente de la aplicación que los necesite.
- **Timeline DB**: es la BBDD que almacena los Tweets y su información asociada. Los Tweets los recibe del Status Content Provider cuando llegan nuevos y se los manda al mismo cuando son solicitados.

- **GAE**: es el servidor que comprime y gestiona las estadísticas explicado en el siguiente apartado.
- **Twitter:** el servicio web Twitter es accedido a través de la API para recibir y publicar contenido del servicio web.

6.5 Servidor

El diseño del servidor puede observarse en la siguiente Ilustración 36 Diseño GAE:

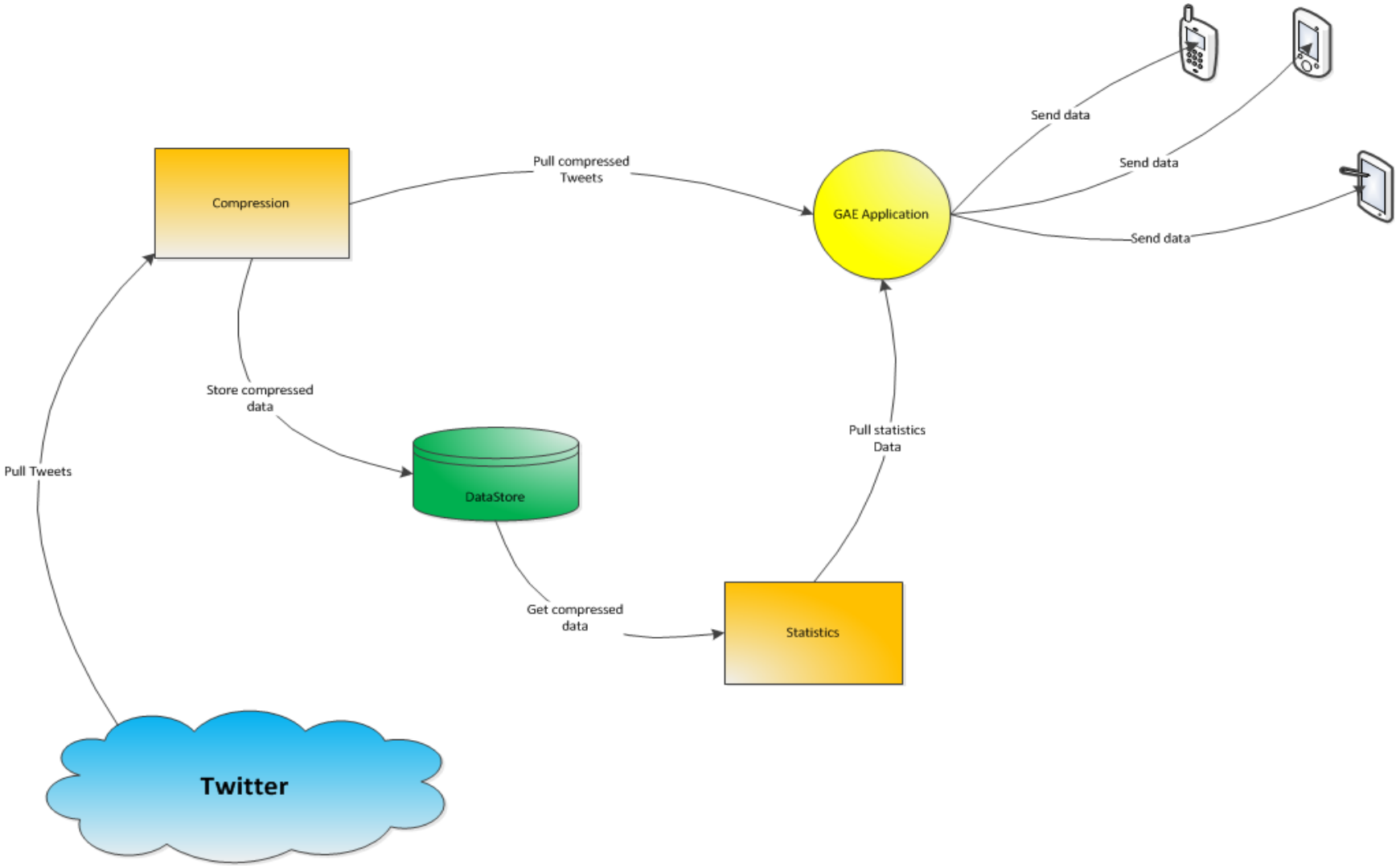

Ilustración 36 Diseño GAE

Aquí se explica cada componente del servidor:

- **Twitter:** es el servicio web que proporciona el contenido de la red social.
- **GAE Application:** actúa como enrutador a los distintos recursos del servidor.
- **Compression:** es uno de los recursos del servidor. Este recibe las credenciales del Smartphone y solicita a Twitter los Tweets en función de estas. Comprime y serializa los Tweets para mandarlos a la aplicación.
- **Statistics:** recibe un identificador de usuario y recoge los datos del mismo para procesarlos en forma de estadísticas al Smartphone. Estos datos también son comprimidos y serializados antes de ser mandados.
- **DataStore:** se encarga de guardar los datos relativos a una petición, que son el usuario, los bytes sin comprimir, los bytes comprimidos y la hora y fecha a la que se produjo la petición. También se encarga de recoger todos los datos de un usuario para que el recurso Statistics pueda mandar los datos estadísticos.
- **Smartphones:** son los dispositivos móviles con la aplicación del cliente instalada que pueden interaccionar con el servidor.

6.6 Interfaz de usuario

En este apartado se definen las interfaces de usuario de la aplicación, que será el medio por el cual el usuario interacciona con el sistema accediendo a sus diferentes funciones. Esta debe ser intuitiva, sencilla y fácil de comprender.

En este caso el servidor carece de interfaz de usuario ya que la única forma de interaccionar con él para el usuario es a través de la aplicación.

La aplicación para el Smartphone debe aprovechar al máximo el espacio reducido de la pantalla intentando ser lo más minimalista posible pero sin dejar de ofrecer información ni opciones.

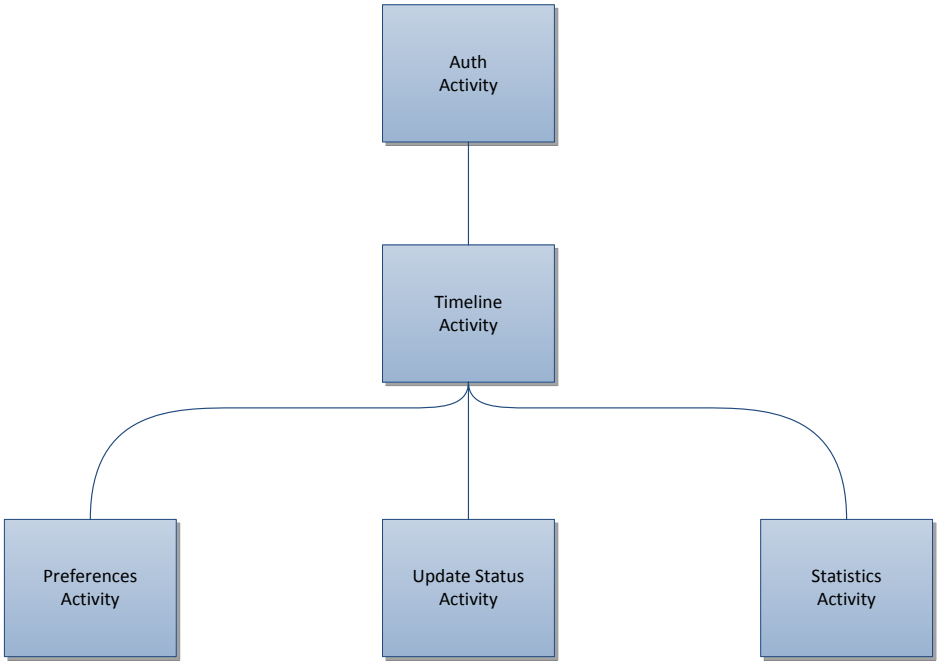

Ilustración 37 Diseño Interfaz App

Como se puede apreciar en la imagen, las pantallas se corresponden con cada uno de los Activities definidos en el diseño de la aplicación Android, y podemos ver también como se navega entre ellos.

A continuación se explica cada interfaz por separado.

6.6.1 Auth Activity

Este Activity sólo se muestra si no existen datos para loguearse. Cuando estos datos son encontrados en la aplicación se pasa directamente al Timeline Activity.

La interfaz es muy sencilla, se muestra un único botón que al ser pulsado da paso a un WebView que accede a Twitter. Desde la web de Twitter se piden los datos de usuario y contraseña y el WebView se cierra y devuelve al Activity las credenciales necesarias.

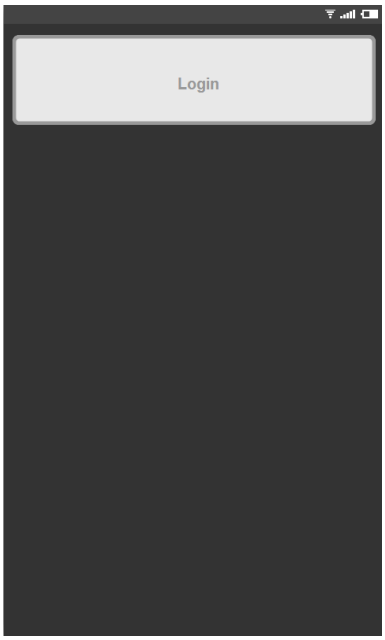

Ilustración 38 Pantalla Auth Activity

6.6.2 Timeline Activity

La interfaz del Timline debe tener un único elemento que a su vez contendrá una lista de elementos. Esta lista de elementos son los Tweets. Cada elemento de la lista debe mostrar a su vez un texto correspondiente al Tweet, la hora de publicación del mismo, el nombre de usuario y la imagen del usuario (avatar).

La disposición de los elementos debe ser la que se observa en la imagen.

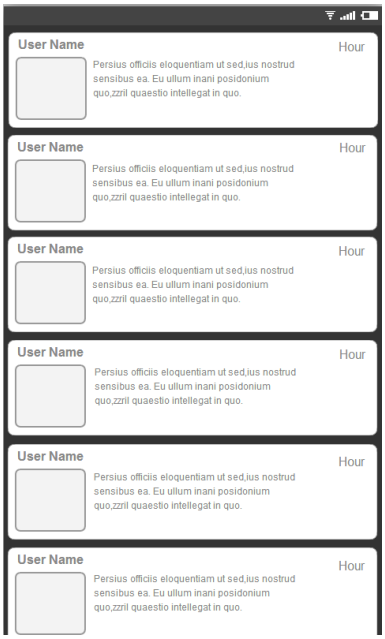

Ilustración 39 Pantalla Timeline Activity

6.6.3 Preferences Activity

A este Activity se llegará a través del menú disponible en Timeline Activity al pulsar la opción Preferences. Esta pantalla mostrará la opción de seleccionar el nivel de compresión, la cual al ser pulsada mostrará las 4 opciones disponibles para comprimir y permitirá seleccionar una. Siempre debe estar una seleccionada.

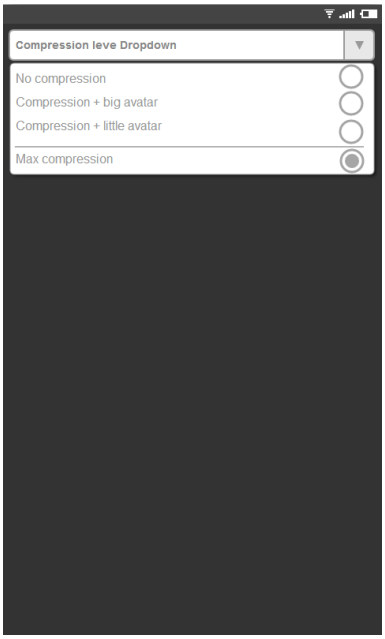

Ilustración 40 Pantalla Preferences Activity

6.6.4 Update Status Activity

A este Activity se llegará a través del menú disponible en el Timeline Activity al pulsar sobre la opción Update Status. Esta pantalla permitirá escribir texto a través de un teclado virtual, mostrará un botón que al ser pulsado actualizará el estado escrito y por último contará con un número que informará de los caracteres restantes a ser escritos.

Ilustración 41 Pantalla Update Status

6.6.5 Statistics Activity

A este Activity se llegará a través del menú disponible en el Timeline Activity al pulsar sobre la opción Statistics. Esta pantalla mostrará dos gráficas, una con los datos comparativos de número de bytes sin comprimir y comprimido, y otra con la tasa de compresión en cada caso. Además se mostrará el número de bytes ahorrados al usar el servicio hasta el momento.

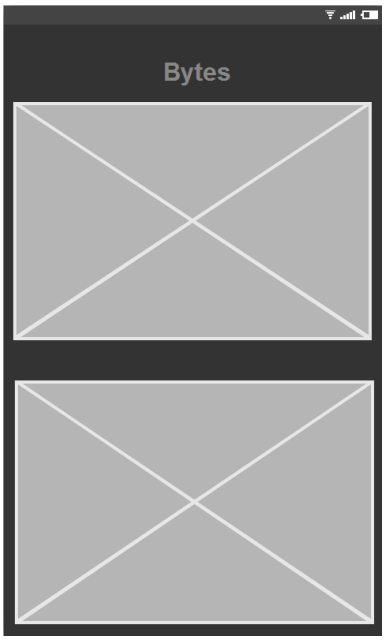

Ilustración 42 Pantalla Statistics Activity

7. DESARROLLO

En esta sección se explica la implementación de este proyecto, tanto de la aplicación móvil como del servidor.

7.1 Aplicación móvil

El desarrollo de la aplicación se realiza siguiendo los esquemas descritos durante el diseño. En esta sección se explica el desarrollo de cada componente del diseño de la aplicación con la inclusión de algunos puntos explicativos más que necesitan ser explicados. Los componentes son los siguientes:

- AsyncTask
- AuthActivity
- TweetezApplication
- TimelineActivity
- UpdaterService
- Cache
- StatusUpdateActivity
- PrefsActivity
- StatisticsActivity
- BootReceiver
- StatusData
- DbHelper
- Interfaces
- AndroidManifest
- XML

7.1.1 AsyncTask

El uso de la clase AsyncTask se da a lo largo de todo el proyecto y por ello se considera necesaria su explicación por separado.

En primer lugar se debe tener en cuenta que todas las operaciones que conciernen al cliente web son realizadas en un hilo distinto del que se ejecuta en la interfaz de usuario. Esta práctica es obligatoria desde la versión 4 de Android y es muy recomendable en cualquier aplicación, con motivo de evitar la ralentización o congelación de la interfaz debido a la ejecución de un proceso de duración variable como son las operaciones de red o de disco.

Esto puede hacerse de distintas formas por medio del lenguaje Java, pero Google recomienda el uso de una clase específica en Android para tal práctica: AsyncTask.

Para hacer uso de esta clase, se crea una subclase donde la vayamos a usar, y esta subclase heredará de AsyncTask, lo que nos permite sobrescribir varios métodos:

- ```
  @Override
  protected String doInBackground(Void... params)
  ```

- ```
  @Override
  protected void onPreExecute()
  ```

- ```
  @Override
  protected void onPostExecute(String result)
  ```

El código que necesitamos que sea ejecutado en segundo plano, esto es las conexiones a la red o las operaciones de disco se incluye en doInBackground(), donde el resto de métodos son usados para otras operaciones relacionadas, pero que se ejecutan en el hilo principal, como puede ser indicar visualmente que la tarea aún se está ejecutando. La clase AsyncTask es usada ampliamente a lo largo de todo el proyecto.

7.1.2 AuthActivity

AuthActivity se encarga de autenticar a un usuario que introduce sus datos de usuario. Esta funcionalidad se codifica en "/src/com/android/tweetez/AuthActivity.java".

Para entender la funcionalidad hay que tener en cuenta que se utilizar objetos que pertenecen a la API de Twitter como son un RequestToken o un ConfigurationBuilder, elementos que sirven para autenticar e identificar a un usuario de Twitter y generar la información necesaria asociada al mismo.

En primer lugar se carga la interfaz de usuario que se encuentra en el XML "/res/layout/main_oatuh.xml" que "inflará" el código necesario para que se muestre la interfaz. Esta mostrará un botón que al ser pulsado iniciará el proceso de autenticación al llamar al método **askOauth()** que sigue el siguiente pseudocódigo:

- Crea un objeto ConfigurationBuilder.
- Se le aplican las credenciales del desarrollador Twitter.
- Genera un objeto Twitter con dichas credenciales.
- AsyncTask:
  - Se solicita un RequestToken a Twitter.
- Extrae del RequestToken la URL para autenticarse.
- Inicia un WebView con la URL generada.

Cuando el usuario ha introducido sus credenciales correctamente la aplicación regresa al mismo Activity, el AuthActivity y se gestiona la información que el **WebView** devuelve:
  ```

- Extrae la información de la URI.
- SI tiene información Y coincide con la url especificada antes:
    - AsyncTask:
        - Se pide un AccessToken con la información extraída.
    - Guarda en disco el AccessToken.
    - Guarda en disco el TokenSecret extraido del AccessToken.

Acto seguido se ejecuta el método **onResume()**, que guardará ciertos datos en un Application, que es explicado en el siguiente punto. Se ejecuta lo siguiente:

- Se comprueba si existe AccessToken guardados.
- SI existe:
    - Extrae el AccessToken y TokenSecret guardados en disco.
    - Genera un objeto ConfigurationBuilder.
    - Le pasa al ConfigurationBuilder todas las credenciales.
    - Genera un objeto Twitter.
    - Guarda el objeto Twitter en la Application TweetezApplication.
    - Lanza el Activity TimelineActivity.
- SI NO existe:
    - Vuelve al inicio.

7.1.3 TweetezApplication

Los Application en Android son utilizados para mantener un estado global de la aplicación, esto es, para tener un punto en común en el que guardar la información necesaria que se usa a lo largo de las distintas clases. Es por esto que el objeto Twitter se guarda en el Application, porque este será usado por otras clases cuando se necesite. Estas clases solo tendrán que pedir al objeto TweetezApplication dicha información. El TweetezApplication se encuentra en "/src/com/android/tweetez/application/TweetezApplication.java".

Las Application sirven además para ejecutar operaciones globales que puedan ser accedidas desde cualquier punto de la aplicación. En el caso de este proyecto servirá para comprobar si el servicio se está ejecutando, y para solicitar Tweets. La solicitud de Tweets se ejecutará desde el Service, explicado más adelante.

Para comprobar el estado del Service, se genera una variable global de tipo boolean que será true cuando esté ejecutándose, y false en caso contrario. Cuando el estado del servicio cambie la clase encargada llamara al Application para informar de que se ha parado y así cambiar la variable. Para conocer el estado, simplemente se solicita dicha variable.

Para recuperar los Tweets se cuenta con un método que solicita los Tweets comprimidos getCompressedHomeTimeline(long), y otro general que decide si solicitarlos a Twitter o a través de este método, llamado fetchStatusUpdates().

El método **getCompressedHomeTimeline(long)** funciona de la siguiente forma:

- Se establece la URL del servidor en un ClientResource.
- Extraen las credenciales de Twitter guardadas.
- Extrae del Application el ID del último Tweet guardado.
- Se hace un POST al servidor con las credenciales y el ID.
- Se guarda la respuesta al POST.
- SI la respuesta es OK:
    - Se hace una petición para descargar los Tweets comprimidos.
    - Son descargados.
    - Los datos descargados se guardan en un objeto a través de una interfaz común a servidor y cliente.
    - Se descomprimen los datos.
    - Se deserializan los datos.
    - Se devuelven los Tweets.
- SI NO se recibe OK por respuesta:
    - Lanza excepción informando.

El método **fetchStatusUpdates()** funciona de la siguiente forma:

- Se guarda el ID del último Tweet recibido.
- Extrae la información del objeto Twitter.
- SI NO existe el objeto Twitter:
    - Sale del método e informa.
- SI existe información:
    - Se comprueba el nivel de compresión seleccionado.
    - SI el nivel es 0:
        - Pide Tweets a Twitter desde el último Tweet.
    - SI NO es 0 el nivel:
        - Se llama a getCompressedTimeline(long) pasándole el ID del último Tweet.
        - SI algo falla al llamar a getCompressedTimeline(long), se pide directamente a Twitter.
    - Se pasan los Tweets recibidos al ContentProvider.

7.1.4 TimelineActivity

Este Activity es principal de la aplicación, ya que es el encargado de mostrar los Tweets, la que permite acceder al resto de componentes de la aplicación y a la que más uso dará el usuario.

La clase se encuentra en "src/com/android/tweetez/TimelineActivity.java" y contiene los métodos que gestionan la caché, que son explicados en el siguiente punto. Además se

encargará de pintar la interfaz con los Tweets, dar acceso al menú y gestionar la lista de Tweets.

El método **setupList()** es el encargado de gestionar la lista de Tweets, y tiene la siguiente funcionalidad:

- Guarda en un Cursor la lista de Tweets y comienza a gestionar el Cursor.
- Comprueba el nivel de compresión seleccionado.
- SI el nivel de compresión es 0, 1 o 2:
  - Carga la interfaz del XML timeline_row.xml que contiene imágenes.
- SI el nivel de compresión es 3:
  - Carga la interfaz del XML timeline_row_no_profile.xml que no contiene imágenes.
- Se carga la lista de Tweets en un Adapter para que los pueda pintar correctamente.
- Se deja de gestionar el Cursor.

Dentro del método setupList() se carga un Adapter en un momento dado, este Adapter permite modificar el comportamiento por defecto para gestionar los Tweets, y sirve para cambiar los datos de la fecha y la URL de la imagen del usuario.

Esto se realiza en el **ViewBinder**:

- SI el ID de entrada coincide con el del campo de la hora a cargar:
  - Se convierte la hora en milisegundos a hora normal.
  - Se carga la nueva hora en la interfaz.
- SI NO coincide:
  - SI coincide con el del campo de la imagen a cargar:
    - Se extrae la URL del cursor.
    - Carga dicha URL en la interfaz llamando a loadBitmap().

Cuando se dice que “Carga dicha URL en la interfaz” se produce un complejo proceso que hace uso de la caché para cargar la imagen en la interfaz, esto es explicado en el apartado de Caché.

A través del método **onOptionsMenuSelected(MenuItem)** se establece la acción a realizar en función de la opción seleccionada al abrir el menú. El pseudocódigo es el siguiente:

- SWITCH MenuItem
- SI ES menuPrefs:
  - Lanza el Activity PrefsActivity.
- SI ES updateStatus:
  - Lanza el Activity StatusUpdateActivity.
- SI ES itemUpdate:

    - Recupera datos de Application.
    - AsyncTask:
        - Solicita lista de Tweets a través de fetchStatusUpdates() en segundo plano.
- SI ES itemStatistics:
    - Lanza el Activity StatisticsActivity.
- SI ES itemToggleService:
    - SI el Service se está ejecutando:
        - Para el Service.
        - Notifica al Application que se ha parado.
    - SI el Service está parado:
        - Inicia el Service.
        - Notifica al Application que se ha iniciado.

Además, dentro de TimelineActivity se encuentra otra subclase, el **TimelineReceiver**, que hereda de BroadCastReceiver y se encarga de ejecutar setupList() cuando el Service actualice la lista de Tweets.

7.1.5 Cache

La lógica de la caché diseñada para ahorrar la acción de descargar las imágenes se encuentra en la clase TimelineActivity. Esta caché se compone de dos sistemas: uno para almacenar en memoria RAM y otro para almacenar en disco.

La lógica de las cachés comienza por su inicialización en el método **onCreate():**

- Recupera las características de la memoria del dispositivo.
- Calcula 1/8 de la memoria disponible en el dispositivo.
- Inicializa la cache de RAM en base a esa cantidad de memoria.
- Se calcula el directorio en el que se almacenará la caché:
    - SI está disponible el almacenamiento externo.
        - Se devuelve el path de dicho almacenamiento.
    - SI NO está disponible el almacenamiento externo.
        - Se devuelve el path del interno.
- AsyncTask:
    - Se inicializa la caché de disco y se establece que comprima las imágenes a una calidad del 80%.
    - Se notifica a los procesos que estuviesen esperando a esta inicialización.

En el método del ViewBinder explicado en TimelineActivity, se llama en un momento dado a **loadBitmap(String, ImageView)**, este método es el que desencadena el proceso de búsqueda y guardado de imágenes en la caché.

- SI la tarea a iniciar coincide con otra iniciada:
  - Se cancela
- SI NO coincide:
  - Se carga la imagen de memoria RAM
  - SI la imagen existe en RAM:
    - Se carga en la interfaz.
  - SI NO existía en RAM:
    - AsyncTask:
      - Busca la imagen en memoria RAM de nuevo.
      - SI no la encuentra en RAM:
        - la busca en memoria de disco
      - SI no lo encuentra en disco:
        - La descarga de Twitter.
      - Añade la imagen a la caché:
        - Añade la imagen a la caché de RAM.
        - Añade la imagen a la caché de disco.
    - Carga la imagen cargada en la interfaz.

7.1.6 UpdaterService

El Service de la aplicación es el encargado de gestionar en segundo plano las peticiones para actualizar la lista de Tweets cada cierto tiempo. El tiempo seleccionado para actualizar los Tweets es cada 900000 milisegundos, que se corresponde con 15 minutos. La clase se encuentra en "/src/com/android/tweetez/service/UpdaterService.java"

El Service ejecuta un nuevo Thread que ejecuta la siguiente lógica:

- MIENTRAS el servicio este en true:
  - Recupera los datos de la Application.
  - Actualiza la lista de estados llamando a fetchStatusUpdates().
  - SI existen nuevos Tweets:
    - Avisa en Broadcast de que hay nuevos Tweets.
  - Duerme el hilo durante 900000 segundos.

7.1.7 StatusUpdateActivity

Este Activity es el encargado de actualizar el estado de usuario o publicar un Tweet. Para ello permite introducir un texto y es avisado de cuantos caracteres le quedan de los 140 disponibles. Una vez superado el límite de 140, un contado cambiará a color rojo para notificar visualmente al usuario.

La clase se encuentra en "/src/com/android/tweetez/ StatusUpdateActivity.java"

La lógica se ejecuta al pulsar el botón de la interfaz que publica el Tweet con el texto escrito.

- AsyncTask:
  - Recuperar datos del Application
  - Generar un estado pasándole el texto escrito.
  - Publicar a través de Twitter el estado.

7.1.8 PrefsActivity

Este Activity se limita a inflar un XML que contiene los elementos del menú de preferencias. La ubicación del mismo es "/src/com/android/tweetez/PrefsActivity.java". Dentro del XML se encuentra una sola opción para elegir entre las opciones disponibles para comprimir.

La estructura del XML es la siguiente:

```
<PreferenceScreen
xmlns:android="http://schemas.android.com/apk/res/android" >

    <ListPreference
        android:defaultValue="3"
        android:entries="@array/compressionLevel"
        android:entryValues="@array/compressionLevelValues"
        android:key="compressionLevel"
        android:summary="Define the compression level"
        android:title="Compression level" />

</PreferenceScreen>
```

En el XML se accede a los elementos de @array, que no es más que la lista de elementos disponibles para seleccionar en esa opción. La estructura de ese XML @array es la siguiente:

```
<resources>

    <string-array name="compressionLevel">
        <item name="0">No compression</item>
        <item name="1">Compression + big avatar</item>
        <item name="2">Compression + little avatar</item>
        <item name="3">Max Compression</item>
    </string-array>
    <string-array name="compressionLevelValues">
        <item name="0">0</item>
        <item name="1">1</item>
        <item name="2">2</item>
        <item name="3">3</item>
    </string-array>


</resources>
```

7.1.9 StatisticsActivity

Este Activity es el encargado de mostrar las estadísticas descargadas del servidor. Nada más iniciarse el Activity se ejecuta un AsyncTask para pedir los datos y al descargarlos actualiza la interfaz mostrándolos en forma de gráficas.

La ubicación de la clase es "/src/com/android/tweetez/StatisticsActivity.java".

La lógica es la siguiente:

- AsyncTask:
  - Se establece la URL del servidor en un ClientResource.
  - Se manda el ID del usuario con un POST.
  - Se recupera la respuesta al POST.
  - SI la respuesta es OK:
    - Se hace un GET para descargas las estadísticas.
- Se descomprimen las estadísticas.
- Se deserializan.
- Se cargan los datos en los objetos GraphView.
- Se añaden dichos objetos a la interfaz.

7.1.10 BootReceiver

El BootReceiver se encarga de iniciar el Service cuando el móvil se enciende. Para ello requiere de un permiso en el AndroidManifest que se analiza más adelante.

Lo único que hace es llamar al BootReceiver que hereda de BroadcastReceiver para iniciar el Service.

7.1.11 StatusData

StatusData es una clase intermedia entre la BBDD y el resto de clases que quieren hacer uso de los datos almacenados en esta. Cuenta con varios métodos para añadir y recuperar datos a la BBDD:

- insertOrIgnore(ContentValues): se encarga de inicializar la BBDD en modo escritura e inserta los valores pasados por parámetros si no hay conflictos.
- getStatusUpdates(): inicializa la BBDD en modo lectura y recupera todos los Tweets disponibles en la misma.
- getLatestStatusCreatedAtTime(): inicializa la BBDD en modo lectura y devuelve el último estado insertado en la BBDD, es decir, el más reciente.

- getLatestStatusId(): Inicializa la BBDD en modo lectura y devuelve el ID del último Tweet insertado en la BBDD.
- getStatusTextById(long id): inicializa la BBDD en modo lectura y devuelve el Tweet que buscamos por su ID pasado por parámetros.

7.1.12 DbHelper

Esta clase se encarga de crear la BBDD y actualizar sus parámetros si se actualiza la versión de la BBDD.

Cuando se llama al objeto en onCreate() este ejecuta una query con los parámetros de creación de la BBDD. La query y su ejecución tienen el siguiente aspecto:

```
String sql = "create table " + TABLE + " (" + C_ID + " int primary
key, " + C_CREATED_AT + " int, " + C_USER + " text, "
                        + C_PROFILE_URL + " text, " + C_TEXT + " text,
" + C_SOURCE + " text)";

            db.execSQL(sql);
```

Donde las claves en mayúsculas representan los parámetros de las columnas a crear.

7.1.13 Interfaces

Los Interfaces en este caso no se refieren a las interfaces de usuario, se refieren a Interfaces de Java. Estas sirven para los objetos que son pasados por la red, y se encuentran exactamente iguales en el servidor. De esta forma al recibir un objeto creado para el proyecto y deserializarlo se puede guardar en un objeto conocido.

Para este proyecto existen tres interfaces:

- **CompDataSerial:** Se trata del interface para el objeto que contiene los datos estadísticos que se procesan en el servidor. Tiene los siguientes atributos:

```
public ArrayList<Integer> compressed;
public ArrayList<Integer> uncompressed;
public ArrayList<String> dateHour;
public ArrayList<Double> compPercentage;

public long maxValue;
```

  Estos parámetros se corresponden con los bytes comprimidos, los descomprimidos, la hora y la tasa de compresión en cada petición realizada. El valor maxValue se corresponde con con el valor más alto en los bytes descomprimidos.

- **CompressedTimeline:** Este interface es necesario para procesar las peticiones POST, GET y DELETE de RESTlet. Tiene estos métodos:

```
@Get
public InputRepresentation retrieve();

@Delete
public void remove();
```

- **StatisticsInterface**: Necesaria también para procesar las peticiones, tiene los siguientes métodos:

```
@Get
public byte [] retrieveData();

@Post
public void tellUser(Representation id);
```

7.1.14 AndroidManifest

El Manifest de una aplicación Android define la estructura general de la misma y es una pieza indispensable para el funcionamiento de la aplicación.

En primer lugar, en el Manifest se definen los permisos de la aplicación. En la aplicación del proyecto son los siguientes:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
```

El primer permiso, WRITE_EXTERNAL_STORAGE es necesario para el funcionamiento de la caché, ya que requiere de guardar imágenes que escribe en la memoria interna del teléfono.

El permiso de INTERNET es necesario para comunicarse tanto con Twitter como con el GAE.

El permiso RECEIVE_BOOT_COMPLETED es necesario para que el servicio de actualización se inicie cuando se arranca el teléfono.

En el Manifest se definen además todos los Applications, Activities y Services usados en la aplicación. Si no se definen en el Manifest estos no funcionarán por no ser reconocidos. Además a cada Activity se le pueden asignar Intent FIlters, que sirven para definir propiedades de los Activities. Es el caso del Activity inicial, el AuthActivity que tiene la siguiente declaración en el Manifest:

```
<activity
      android:name="AuthActivity"
      android:label="Tweetez" >
      <intent-filter>
          <action android:name="android.intent.action.MAIN" />
```

```
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
        <intent-filter>
            <action android:name="android.intent.action.VIEW" />

            <category android:name="android.intent.category.DEFAULT" />
            <category android:name="android.intent.category.BROWSABLE" />

            <data
                android:host="tweeter"
                android:scheme="callback" />
        </intent-filter>
    </activity>
```

En los Intent Filter se define el action.MAIN para indicar que es el primer Activity que debe ejecutarse al arrancar la aplicación. Pero además se le asigna el action.VIEW así como category.DEFAULT y category.BROWSABLE para que pueda gestionar el WebView que abre posteriormente.

Además de definir el resto de Activities y el Service de forma normal, se define el BootReceiver con el siguiente código:

```
<receiver android:name=".util.BootReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

7.1.15 XML

A lo largo del proyecto se hace uso de otros archivos XML además del Manifest y las intefaces que son cargadas a través de los XML correspondientes.

Se hace uso de otro XML, el status_menu.xml que contiene el menú que se abre en el TimelineActivity para acceder a las acciones explicadas en el mismo. Dicho XML tiene la siguiente estructura:

```
<item
    android:id="@+id/menuPrefs"
    android:icon="@drawable/ic_menu_moreoverflow_normal_holo_dark"
    android:title="@string/MenuTitlePrefs">
</item>
<item
    android:id="@+id/itemToggleService"
    android:icon="@drawable/ic_menu_play_clip"
    android:title="@string/MenuServiceToggle">
</item>
<item
    android:id="@+id/itemUpdateStatus"
    android:icon="@drawable/ic_menu_compose"
    android:title="@string/MenuUpdateStatus">
```

```
</item>
<item
    android:id="@+id/itemStatistics"
    android:icon="@drawable/ic_menu_info_details"
    android:title="@string/MenuStatistics">
</item>
<item
    android:id="@+id/itemUpdate"
    android:icon="@drawable/ic_menu_refresh"
    android:title="@string/TimelineUpdate">
</item>
<item
    android:id="@+id/itemRestore"
    android:icon="@drawable/ic_menu_refresh"
    android:title="Restore Position">
</item>
```

En este se especifican los elementos del menú y sus asociaciones en el Activity para que puedan ser llamados.

El otro XML usado se trata del archivo strings.xml que contiene referencias a textos para escribir dichas referencias en el código en vez de escribir directamente el texto. Esto es para evitar tener que cambiar un texto en muchos lugares si se produce algún cambio o para poder asignar varios idiomas a una aplicación.

7.1.16 Bibliotecas

A lo largo del proyecto se hace uso de varias APIs para posibilitar el funcionamiento. Dichas APIs se encuentran en forma de archivos .jar en el directorio “/libs/” y son las siguientes:

- Commons-compress-1.4.1.jar: esta biblioteca es usada para los algoritmos de compresión de GZip.
- GraphView-3.0.jar: es la biblioteca usada para representar y organizar los datos estadísticos en forma de gráficas.
- Org.restlet.jar: es la biblioteca usada para las comunicaciones con el servidor usando el servicio RESTlet.
- Twitter4j-core-3.0.2.jar: permite acceder a las llamadas de Twitter de una forma más sencilla.

7.2 Servidor

El desarrollo del servidor se realiza siguiendo los esquemas descritos durante el diseño. En esta sección se explica el desarrollo de cada componente del servidor. Los componentes son los siguientes:

- TweetezGaeApplication
- Compression
- Statistics
- DataStore
- Interfaces
- Bibliotecas

7.2.1 TweetezGaeApplication

De la misma forma que funciona el Application en la aplicación de Android, existe en el servidor del GAE por utilizar las librerías de RESTlet. El Application sirve como punto central para el resto de la aplicación y en este caso contendrá los siguientes elementos:

```
private String tokens;

private long user_id;

@Override
public Restlet createInboundRoot() {
      Router router = new Router(getContext());

      router.attachDefault(new Directory(getContext(),
      "war:///"));
      router.attach("/compress", Compression.class);
      router.attach("/statistics", Statistics.class);

      return router;
}
```

Las variables globales son usadas para el resto de la aplicación. Tokens se corresponde con las credenciales que recibe el servidor para pedir la lista de Tweets. El user_id es usado para identificar un usuario a la hora de solicitar las estadísticas.

El método **createInboundRoot()** sirve de enrutador para los demás componentes de la aplicación que son accedidos por los clientes. En este caso se crea un Router, y se le añaden las dos aplicaciones (Compression y Statistics) asignándoles a cada una una URI.

7.2.2 Compression

Es la clase encargada de pedir la lista de Tweets, comprimirlo y mandarlos al cliente. Contiene tres métodos asociados a POST, GET y DELETE, que son receive(Representation), retrieve() y remove() respectivamente.

La lógica de receive(Representation) es la siguiente:

- Crea una nueva Application y le asigna los datos existentes en ese momento.
- Asigna a la variable tokens del Application el texto recibido.

La lógica de retrieve() es la siguiente:

- Se crea un array de bytes vacío.
- Se reciben las credenciales del cliente con el ID del último Tweet.
- Se procesan los datos para ser usados.
- Se crea un ConfigurationBuilder con las credenciales.
- Se genera un objeto Twitter a través del ConfigurationBuilder.
- SI el ID del último Tweet es 0:
  - Se solicitan los últimos 60 Tweets del usuario.
- SI NO es 0:
  - Se solicitan los últimos 60 Tweets hasta el Tweet indicado.
- Se serializa la lista de Tweets recibidos.
- Se comprimen los datos serializados.
- Se guardan los datos asociados al a petición a través del DataStore.
- Se devuelven los datos serializados.

La lógica de remove() es la siguiente:

- Se recuperan los datos del Application.
- Se establecen a “vacio” los tokens del Application.

7.2.3 Statistics

La clase Statistics del servidor es la encargada de procesar las estadísticas para mandarlas al cliente. Cuenta con dos métodos asociados a POST y GET, que son tellUser(Representation) y retrieveData() respectivamente.

La lógica de tellUser(Representation) es la siguiente:

- Se recuperan los datos del Application.
- Se establece el id pasado por parámetros al parámetro user_id del Application.

La lógica del retrieveData() es la siguiente:

- Se recuperan los datos del Application.
- Se recuperan los datos estadísticos del DataStore.
- Se calcula el valor más alto de dichas estadísticas.
- Se guarda dicho valor en la interface CompDataSerial.
- Se calcula la tasa de compresión para cada petición.
- Se guardan los valores en un array que se guarda en la interface CompDataSerial.
- Se serializan la interface con todos los datos.
- Se comprimen los datos.
- Se devuelve la interfaz CompDataSerial comprimida.

7.2.4 DataStore

Esta clase es la encargada de recuperar y guardar los datos almacenados en el Google App Datastore. Consta de dos métodos, uno para guardar los datos y otro para recuperarlos.

El método para guardar los datos es **saveCompressionLevel(long,int,int,Calendar)** y tiene la siguiente lógica:

- Se guarda en un String la hora recibida por el Calendar procesada.
- Se crea un nuevo objeto Entity.
- Se cargan los datos recibidos por parámetros en el Entity.
- Se hacen persistentes en el servidor.
- Se devuelve true si todo OK.

El método para recuperar los datos es **getStatistics(long)** y tiene la siguiente lógica:

- Se crea un objeto de la interface CompDataSerial.
- Se crea una nueva Query para buscar en la tabla de datos.
- Se añade un filtro con el usuario pasado por parámetros.
- Se ejecuta la Query.
- Se guardan los datos recibidos por la Query en el objeto CompDataSerial.
- Se devuelve el objeto con los datos conseguidos.

7.2.5 Interfaces

Al igual que en la aplicación Android, el servidor cuenta con varios Interfaces para que la comunicación por red sea satisfactoria. Estos Interfaces son exactamente los mismos explicados en el apartado 6.1.13 Interfaces.

7.2.6 Bibliotecas

Las bibliotecas usadas en el servidor son también las mismas que usa la aplicación Android a excepción de GraphView, ya que esta sirve para pintar las estadísticas, función que no se acomete en el servidor. El resto de bibliotecas usadas son las mismas que en el apartado 6.1.16 Bibliotecas.

8. PRUEBAS DE EVALUACIÓN

En esta sección se realizan las pruebas pertinentes para comprobar el funcionamiento del sistema desarrollado. En estas pruebas se usará un servidor y Smartphone cuyas especificaciones hardware se pueden observar en las siguientes tablas:

Tabla 28 Hardware de prueba

Modelo	Samsung Galaxy S I-9000
CPU	Cortex A-8 1 Ghz
Memoria	512 Mb RAM
Conectividad	HSDPA

Los recursos reales utilizados en el servidor son difíciles de calcular debido a que se trata de un sistema distribuido que asigna los recursos en función de las necesidades de la aplicación en cada momento, luego tampoco dispone de unos recursos fijos si no que estos son dinámicos y adaptados a la carga de trabajo del servidor.

Los datos a calcular en el proyecto son los siguientes:

- Tiempo en actualizar un estado de 90 caracteres.
- Tiempo en recuperar un Timeline completo (60 Tweets) a través de Twitter.
- Tiempo en recuperar un Timeline completo (60 Tweets) a través del GAE.
- Tiempo en recuperar las estadísticas.
- Tasa de compresión media en el servidor.

Los tiempos de las pruebas que requieren conexiones a la red se realizarán en dos casos. La primera sobre red Wi-FI con conexión de 10 Mbps y la segunda sobre la red 3G del Smartphone, en una situación de cobertura HSDPA a 7.2 Mbps aproximadamente.

Las pruebas se calculan haciendo uso de la función de Java System.currentTimeMillis() que devuelve la hora al momento de ejecutarlo en milisegundos. Con esto se hacen 10 tomas de tiempo de cada caso y se calcula la media, desviación típica y se muestra el máximo y mínimo valor. Todos los tiempos mostrados están en milisegundos

8.1 Actualización de estado

Los cálculos se hacen dentro de la propia AsyncTask cuando ejecuta la llamada principal a la red de Twitter. Los datos conseguidos son los siguientes:

Tabla 29 Actualización estado Wi-fi

Sobre red WI-FI	
Número de pruebas	10
Mínimo	731 ms
Máximo	1151 ms
Media	832,4 ms
Desviación Típica	124,66 ms

Tabla 30 Actualización estado 3G

Sobre red 3G	
Número de pruebas	10
Mínimo	698 ms
Máximo	1056 ms
Media	781,1 ms
Desviación Típica	112,02 ms

Los tiempos en este caso son muy similares, siendo algo superiores en la red Wi-fi. Esto probablemente es debido a la velocidad de subida de la red 3G, superior a la de la Wi-fi en este caso. Sin embargo teniendo en cuenta la desviación típica en ambas se podría decir que los tiempos son iguales para ambos casos.

8.2 Recuperación de Timeline (Twitter)

Esta prueba se realiza recuperando 60 Tweets directamente desde Twitter sin pasar por el GAE. La toma de tiempos se hace igualmente dentro de la AsyncTask.

Los datos obtenidos son los siguientes:

Tabla 31 Recuperación Timeline Wi-fi

Sobre red WI-FI	
Número de pruebas	10
Mínimo	1823 ms
Máximo	2249 ms
Media	1936,6 ms
Desviación Típica	136,75 ms

Tabla 32 Recuperación Timeline 3G

Sobre red 3G	
Número de pruebas	10
Mínimo	4818 ms
Máximo	5316 ms
Media	4977,6 ms
Desviación Típica	150,65 ms

En este caso, con prácticamente las mismas desviaciones típicas, se aprecia claramente como el uso de la red 3G para descargar la lista de Tweets directamente desde Twitter es mucho más lenta, llegando de media a tardar más del doble.

8.3 Recuperación de Timeline (GAE)

La prueba es la misma que la anterior aunque en esta ocasión se hace uso del Proxy y de la compresión de datos para recibir los 60 Tweets.

Los datos conseguidos son los siguientes:

Tabla 33 Recuperación Timeline GAE Wi-fi

Sobre red WI-FI	
Número de pruebas	10
Mínimo	2660 ms
Máximo	3115 ms
Media	2937,7 ms
Desviación Típica	130,69 ms

Tabla 34 Recuperación Timeline GAE 3G

Sobre red 3G	
Número de pruebas	10
Mínimo	2606 ms
Máximo	6064 ms
Media	3740 ms
Desviación Típica	1365 ms

En este caso, respecto a pedir los Tweets directamente a Twitter, se observa que la red 3G sigue siendo más lenta. Sin embargo la desviación en el caso del 3G se dispara, convirtiendo los datos en mucho menos fiables.

8.4 Recuperación de estadísticas

Esta prueba, igual que las anteriores, toma los tiempos dentro de la AsyncTask.

Los datos conseguidos son los siguientes:

Tabla 35 Recuperación estadísticas Wi-fi

Sobre red WI-FI	
Número de pruebas	10
Mínimo	877 ms
Máximo	2828 ms
Media	1352,8 ms
Desviación Típica	588,09 ms

Tabla 36 Recuperación estadísticas 3G

Sobre red 3G	
Número de pruebas	10
Mínimo	936 ms
Máximo	6433 ms
Media	2258 ms
Desviación Típica	1763,22 ms

Al realizar la operación de petición de estadísticas al GAE ocurre algo muy similar al pedir la lista de Tweets al GAE. Los tiempos son superiores en la red 3G y la desviación es también muy alta haciendo los datos de la red 3G menos fiables.

8.5 Tasa de compresión en el servidor

Los datos reunidos hasta el momento en el servidor permiten hacer un buen estudio acerca de la capacidad media de compresión del servidor. Para ello se toman en consideración todos los

valores, sin diferenciar entre usuarios y se calcula el máximo, el mínimo, la varianza y la media de las tasas de compresión conseguidas.

Los resultados obtenidos son los siguientes:

Tabla 37 Tasa de compresión en servidor

Tasa de compresión del servidor	
Número de pruebas	90
Mínimo	38%
Máximo	85%
Media	63%
Desviación Típica	23%

Los datos relativos a la compresión y descompresión de datos fueron analizados en la sección 3 ALGORITMOS DE COMPRESIÓN.

8.6 Conclusiones de las pruebas de evaluación

A la vista de los resultados se puede decir que el uso del GAE es satisfactorio, ahorrando una importante cantidad de bytes al usar el GAE. Los tiempos al usar 3G para descargar la lista de Tweets son muy similares a los de usar la red Wi-fi, si bien algo más inestables en cuanto al tiempo que tardan.

En general, la red 3G es algo más lenta como era de esperar, pero sorprende la inestabilidad de los datos de la misma, llegando a tardar hasta 4 veces más en completar una petición que otras veces. Esto hace que al usar 3G los tiempos de espera puedan ser muy bajos o muy altos, probablemente debido a la disponibilidad de la red en ese momento, la cobertura y la influencia del resto de aplicaciones del Smartphone que hacen uso de la red.

Las tasas de compresión conseguidas por el servidor son excelentes, ahorrando de media más de un 60%, con lo que se consiguen descargar más de la mitad de datos que se descargarían normalmente. Los datos de compresión sin embargo también varían bastante con casos en los que se consiguen tasas cercanas al 90% y tasas inferiores al 40%. Los mejores casos son aquellos en los que mayor número de Tweets se descarga, obteniendo los peores resultados al descargar pocos Tweets, por lo que para optimizar al máximo el uso del servidor lo ideal sería que el usuario actualizase cada periodos largos de tiempo o incluso sólo en las ocasiones que lo necesite, de forma manual. Con esto se conseguirían prácticamente siempre tasas muy altas de compresión.

9. CONCLUSIONES Y LÍNEAS FUTURAS

9.1 Conclusiones

El Trabajo de Fin de Grado ha conseguido los objetivos marcados para el mismo, consiguiendo una implementación básica de un cliente Twitter de manera totalmente funcional e intuitiva, consiguiendo una experiencia similar a la del cliente oficial, salvando las funcionalidades no desarrolladas en este proyecto.

La inclusión del servidor actuando de Proxy, comprimiendo los datos y almacenando los datos estadísticos es todo un acierto ya que consigue el objetivo básico de ahorrar bytes, pero además permite descargar parte del trabajo de algunas funcionalidades del Smartphone en el servidor. Esto, según el tipo de tarea podría ahorrar batería y reducir los tiempos de carga de algunas funcionalidades, especialmente si se han de procesar grandes datos dando un pequeño resultado, como el caso de las estadísticas. Por ello probablemente cuanto más uso en el tiempo se haga del servidor, mayor será su aprovechamiento.

En cuanto al rendimiento ofrecido por el servidor, este en general aumenta los tiempos respecto al uso directo de Twitter, motivado por el mayor recorrido que deben realizar los datos, que deben ser comprimidos y serializados durante el proceso. Sin embargo esto se compensa en cierta medida con la menor descarga de datos, aunque al tratarse de cantidades pequeñas, el ahorro en tiempo al descargar no es muy alto.

A las conclusiones de las tasas de compresión conseguidas por el servidor debe añadirse el hecho de que no se cuenta con el posible ahorro asociado a las imágenes, ya que si bien estas no pasan por el Proxy para ser comprimidas porque se puede seleccionar un tamaño suficientemente pequeño, sí que se puede seleccionar distintas opciones respecto a las imágenes. El uso de datos varía notablemente cuando no se usan imágenes que cuando si se usan, y además se pueden descargar imágenes de alta o baja calidad, lo que a pesar del uso de la caché, determinará descargar más o menos datos.

Por último cabe concluir que el impacto en la batería puede ser notable según la opción de compresión seleccionada. Al no pasar por el Proxy los tiempos de descarga son algo mejores, sin embargo la cantidad de datos descargada es mayor, pero el mayor impacto es al seleccionar la carga o no de imágenes ya que la opción en la que no se muestran los avatares evita el uso de la caché y la descarga de imágenes, que supone una de las mayores cargas de computación para la aplicación móvil.

9.2 Líneas futuras

En cuanto a las **líneas futuras**, se plantean las siguientes:

- Completar la funcionalidad del cliente Twitter, añadiendo gestión de mensajes privados, favoritos, interacción con los Tweets, etc.…
- Aprovechar el servidor con las nuevas funcionalidades, comprimiendo otras listas de Tweets que se pueden solicitar a Twitter, como la lista de menciones o mensajes directos.
- Optimizar el servidor utilizando algún algoritmo de mayor rendimiento como el LZO. Para ello habría que abandonar Google App Engine y buscar una opción que ofrezca una escalabilidad parecida con la opción de incluir código C o C++.
- Traspasar los cálculos más intensivos al servidor y añadir y optimizar funcionalidades de cara al ahorro de datos, como una cache en el servidor.
- Optimizar el uso de la batería en el dispositivo al usar la aplicación.

Aunando estas líneas futuras se podría conseguir una aplicación completa que competiría de manera directa con el cliente oficial de Twitter al ofrecer sus mismas funcionalidades pero ofreciendo otras extra como el ahorro de datos y otras funciones que podrían añadirse al servidor. Entre esas nuevas funcionalidades se podría encontrar la gestión inteligente de Tweets o filtros, incluso ofrecer una determinada experiencia en función de factores meteorológicos o definidos por el usuario que podrían ser leídos por el servidor, este procesaría una lista de Tweets inteligente y la mandaría al cliente.

Por último debería optimizarse al máximo el uso de batería en el dispositivo, haciendo uso de tecnologías como las notificaciones Push para evitar estar consultando el servidor para ver si hay nuevos Tweets.

10. PRESUPUESTO

Para realizar el cálculo del presupuesto del proyecto se han tomado en consideración diferentes aspectos, en primer lugar los requisitos técnicos y de hardware:

Tabla 38 Presupuesto Hardware

Elemento	Coste unitario	Nº Unidades	Total
Ordenador Personal para el desarrollo	840,00 €	1	840,00 €
Google App Engine	0 €	1	0 €
Coste Total		840 €	

Tras los costes hardware se deben tener en cuenta los recursos de software utilizados:

Tabla 39 Presupuesto Software

Elemento	Coste unitario	Nº Unidades	Total
Licencia Windows 7	164,00 €	1	164,00 €
Licencia Microsoft Office 2010	199,00 €	1	199,00 €
Software libre	0 €	1	0 €
Coste total			363,00 €

El tiempo de vida estimado de los recursos hardware, a donde los recursos software están adscritos, es de 30 meses de duración, por lo que se ajustan los costes amortizados, en base al a duración del proyecto, de 8 meses:

Tabla 40 Presupuesto amortizado

Elemento	Coste completo	Coste mensual	Coste amortizado
Amortización HW	840,00 €	28,00 €	224,00 €
Amortización SF	363,00 e	12,10 €	96,80 €
Coste total			320,8 €

Se desglosan también los gastos de personal, con las horas invertidas y el coste de cada hora:

Tabla 41 Presupuesto Personal

Elemento	Coste hora	Coste mes	Horas totales	Coste proyecto
Becario	5 €	400,00 €	480	2400,00 €
Personal investigador	11,25 €	1800,00 €	300	3375,00 €
Personal investigador	11,25	1800,00 €	300	3375,00 €
Coste total				9150,00 €

Se deben considerar además los gastos en el transporte:

Tabla 42 Presupuesto transporte

Medio de transporte	Tipo de gasolina	Coste por litro	Coste mensual	Coste total
Coche	Gasolina 95	1,40 €/L	110,00 €	880,00 €

Así pues, uniendo todos los costes anteriores, se consigue como resultado el coste de desarrollo del proyecto, que ha sido:

Tabla 43 Presupuesto final

Elemento	Total
Hardware	224,00 €
Software	96,80 €
Personal	9150,00 €
Medio de transporte	880,00 €
Total	10350,8 €

11. BIBLIOGRAFÍA

1. **Taylor, Chris.** http://mashable.com. *Smartphone Sales Overtake PCs for the First Time [STUDY].* [En línea] http://mashable.com/2012/02/03/smartphone-sales-overtake-pcs/.

2. **Sierras, Cedric.** ¿Qué significan las letras G, E, 3G, H y H+ que muestra mi móvil? [En línea] http://smartphonesworld.es/que-significan-las-letras-g-e-3g-h-y-h-que-muestra-mi-movil/.

3. El número de usuarios de Twitter sobrepasa los 500 millones. [En línea] http://www.elperiodico.com/es/noticias/sociedad/numero-usuarios-twitter-sobrepasa-los-500-millones-2168142.

4. abc.es. *whatsapp-salva-cuatro-montaneras.* [En línea] http://www.abc.es/local-madrid/20121229/abci-whatsapp-salva-cuatro-montaneras-201212291257.html.

5. Apple.com. [En línea] apple.com.

6. **Marko Gargenta.** Marakana. [En línea] http://ofps.oreilly.com/titles/9781449390501.

7. Windows Phone. [En línea] http://www.windowsphone.com/es-es.

8. **Microsoft.** Visual Studio. [En línea] http://www.microsoft.com/visualstudio/eng/downloads.

9. Windows Phone Developers. [En línea] http://dev.windowsphone.com/en-us.

10. **Microsoft.** MSDN. [En línea] http://msdn.microsoft.com/es-es/windowsphone/gg598291.

11. **Wikipedia.** Wikipedia. [En línea] http://es.wikipedia.org/wiki/BlackBerry.

12. **Ruiz, Francisco.** Culturacion.com. [En línea] http://culturacion.com/2012/05/blackberry-sistema-operativo-movil-de-rim/.

13. **Gomez, Diego.** Dos ideas. [En línea] http://www.dosideas.com/noticias/java/332-principios-de-rest.html.

14. **J. Louvel, T. Templer, T. Boil.** *Restlet in Action: Developing RESTful web APIs in Java.* s.l. : Manning, 2012.

15. **Apache.** Apache Axis. [En línea] http://axis.apache.org/axis/.

16. **Curbera, F., y otros.** *Unraveling the Web services web: an introduction to SOAP, WSDL, and UDDI.* 2002. 1089-7801.

17. **Etayo, Esteban.** estebanetayo.es. *WebServices: SOAP vs REST: ¿Cual usar en cada caso?* [En línea] http://www.estebanetayo.es/2011/11/10/webservices-soap-vs-rest-%C2%BFcual-usar-en-cada-caso/.

18. **Carla Lorena Villena Rivera, Juan Timoteo Ponce, Marines Lopez Soliz, Roger Humberto Unoja, Yenny Perez Cervantes.** uagrm-components-design.googlecode.com. [En línea] http://uagrm-components-design.googlecode.com/svn-history/r104/trunk/module10/articles/articulo.doc.

19. Twitter Developers. [En línea] https://dev.twitter.com/docs/twitter-libraries.

20. Twitter4j. [En línea] http://twitter4j.org/en/index.html.

21. **Datos, Agencia Española de Protección de.** http://www.agpd.es. [En línea] http://www.agpd.es/portalwebAGPD/canaldocumentacion/publicaciones/common/Guias/GUIA_CIUDADANO_OK.pdf.

22. —. www.agpd.es. [En línea] http://www.agpd.es/portalwebAGPD/canaldocumentacion/publicaciones/common/pdfs/guia_responsable_ficheros.pdf.

23. GitHub. [En línea] https://github.com/ning/jvm-compressor-benchmark.

24. Twitter Dev. [En línea] https://dev.twitter.com/.

12. MANUALES

En esta sección se incluyen los manuales de instalación y puesta a punto del proyecto así como los manuales de usuario necesarios para utilizar la aplicación.

12.1 Manual de instalación del servidor

La instalación del servidor para utilizarlo como Proxy es muy sencilla debido a que usaremos Google App Engine. Esto hace que no tengamos que configurar el servidor, ni instalar ningún tipo de tecnología que soporte Java, ni bases de datos. La instalación del servidor se simplifica en hacer un "deploy" de la misma a través de nuestra cuenta de Google con el plugin para Eclipse de Google.

Por tanto se explica en primer lugar como instalar el plugin de Google en Eclipse.

Eclipse se descarga desde su página oficial de manera totalmente gratuita, eclipse.org, y entras las versiones que podemos elegir para descargar debemos seleccionar la "Classic" o "Java Developers".

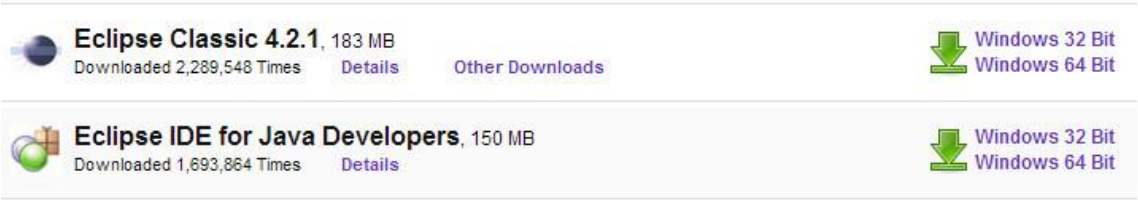

Ilustración 43 Manual servidor, descarga de eclipse

Una vez descargado y arrancado (no requiere instalación), seleccionaremos el menú Help -> Install New Software… Y en la nueva ventana pulsaremos el botón de Add…

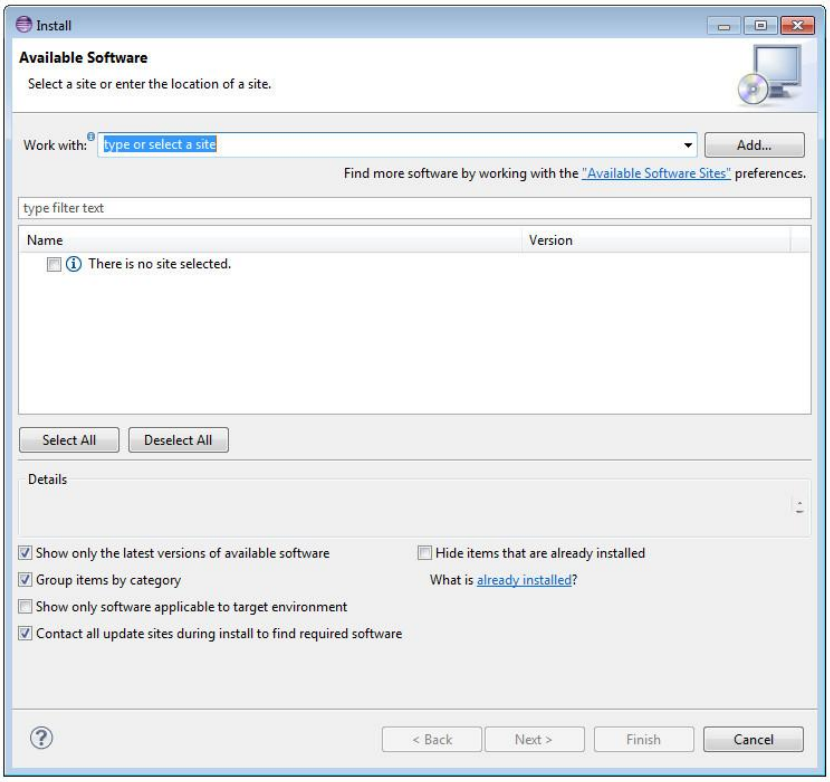

Ilustración 44 Manual Servidor, añadir repositorio

En el campo Name: pondremos Google Plugin, mientras que en el campo Location: debemos introducir la URL del plugin correspondiente a nuestra versión de Eclipse. Dicha URL puede consultarse en el siguiente link:

https://developers.google.com/eclipse/docs/install-eclipse-4.2

Este link incluye además las instrucciones de instalación en inglés.

Ahora debemos seleccionar todas las opciones disponibles a través de esta URL para instalar las herramientas necesarias para interactuar con Google App Engine a través de Eclipse. Al seleccionar todas las opciones instalaremos además las herramientas necesarias para el desarrollo de la plataforma Android.

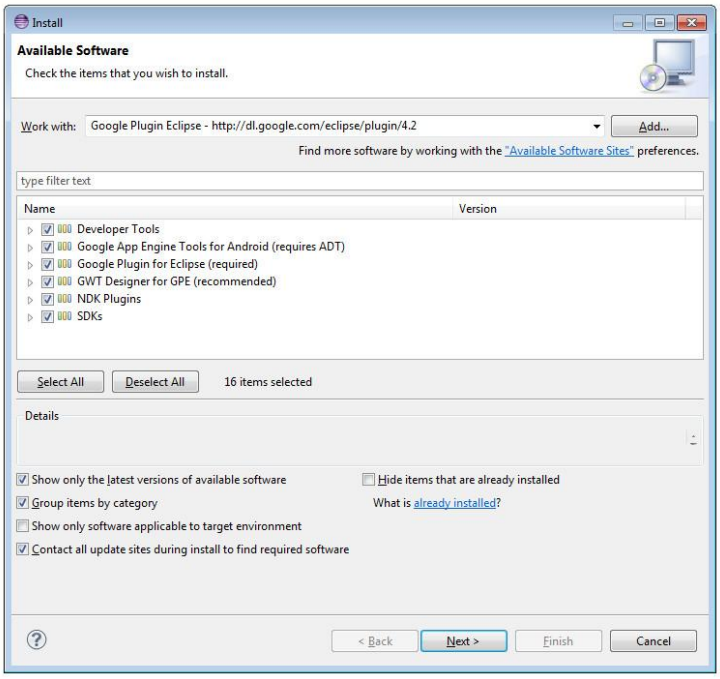

Ilustración 45 Manual Servidor, instalación de paquetes

Mientras se descargan los paquetes, debemos crear una nueva aplicación en la web de App Engine desde el siguiente enlace:

https://appengine.google.com/start/createapp

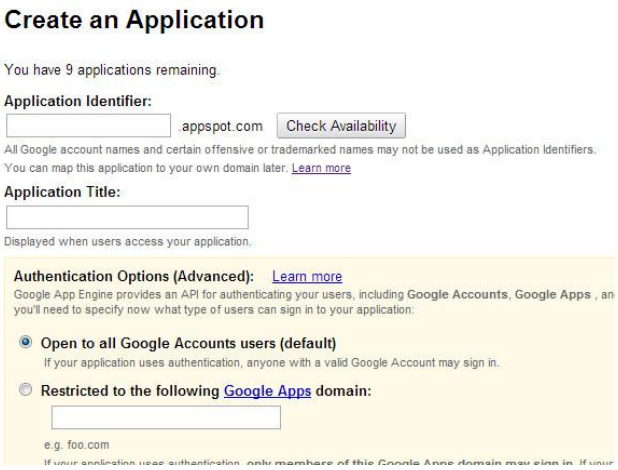

Ilustración 46 Manual Servidor, crear aplicación GAE

Debemos asignar un nombre a nuestra aplicación, y seleccionar un título para la misma. Con esto tendríamos creada nuestra aplicación en Google App Engine.

Tras la descarga e instalación de los paquetes tendremos disponible un nuevo botón en la barra de herramientas de Eclipse con distintas opciones entre las que se encuentra la opción “Deploy to App Engine”.

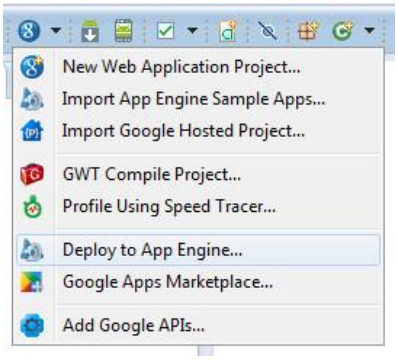

Ilustración 47 Manual Servidor, Deploy to App Engine...

En la siguiente pantalla debemos seleccionar el proyecto a desplegar, que deberemos importar en los proyectos de Eclipse desde File -> Import y luego Existing Projects to Workspace. De esta forma podremos seleccionarlo.

Tan solo restaría pulsar en el botón Deploy y esperar mientras la aplicación se despliega.

12.2 Manual de instalación de la aplicación

Al igual que en la instalación del servidor, para Android utilizaremos el entorno de desarrollo Eclipse, donde podemos instalar las herramientas necesarias para el desarrollo de la aplicación como se explicó en el apartado del servidor.

Ahora seleccionaremos el Android SDK Manager que aparecerá en la barra de herramientas de Eclipse:

Ilustración 48 Manual Android, SDK Manager.

En la siguiente pantalla debemos seleccionar los componentes de Tools y Android 4.2 (API 17) e instalaremos dichos paquetes.

Una vez instaladas las herramientas podremos importar el proyecto en Eclipse haciendo uso del menú File -> Import y luego seleccionando Android -> Existing Android Code Into Workspace.

12.3 Manual de usuario

En este proyecto el servidor es un subsistema completamente transparente al usuario por lo que el manual de usuario explicará el funcionamiento de la aplicación móvil y como esta puede interaccionar con el servidor, pero en ningún caso el usuario podrá ver o modificar el servidor sin que sea a través de la aplicación.

En primer lugar, para arrancar la aplicación debemos acceder a la lista de aplicaciones de nuestro terminal y buscar la aplicación Tweetez.

La primera vez que iniciamos la aplicación nos mostrará la siguiente pantalla:

Ilustración 49 Screenshot 1

Al pulsar el botón de Authenticate with Twitter accederemos al navegador web que nos redirigirá a Twitter donde tendremos que introducir nuestros datos.

Ilustración 50 Screenshot 2

Tras introducir correctamente nuestros datos de cuenta en Twitter accederemos a la pantalla del Timeline, vacío al ser la primera vez, desde el cual podremos acceder al menú de opciones pulsando la tecla de menú en nuestro dispositivo:

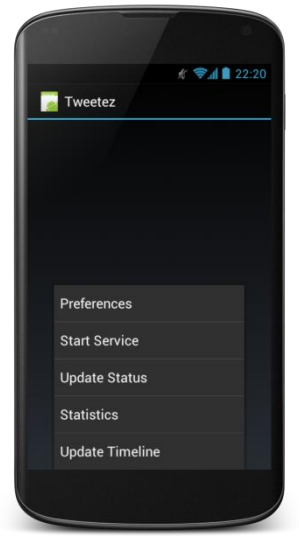

Ilustración 51 Screenshot 3

En primer lugar accederemos a las preferencias pulsando el botón Preferences del menú de opciones, lo que nos lleva al menú de preferencias, donde al seleccionar la única opción Compression level, se mostrarán las opciones disponibles:

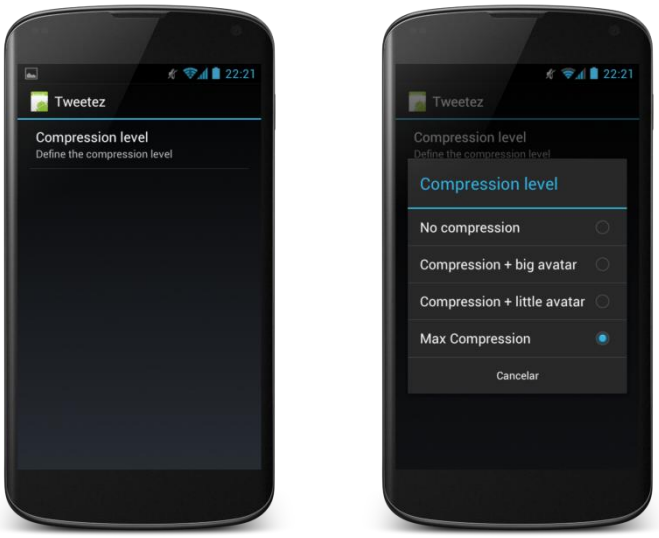

Ilustración 52 Screenshot 4 y 5

Seleccionaremos la opción No compression en primer lugar. Al volver hacia atrás mediante el botón de Android, volveremos a pulsar el botón de menú, y en esta ocasión seleccionaremos la opción Update Timeline, que accederá esta vez a Twitter de manera directa. Mientras se ejecuta la descarga de Tweets se informará mediante una imagen. Cuando se han descargado los Tweets, se mostrarán en la pantalla, junto con los avatares de los respectivos autores de los Tweets.

Ilustración 53 Screenshot 6 y 7

Esta es la vista general que tendremos de los Tweets. Si seleccionamos la opción No compression accederemos a una vista ligeramente distinta en la que no se muestran los avatares:

Ilustración 54 Screenshot 8

Desde el menú de opciones del Timeline, podremos seleccionar la opción Start Service, lo que inicia el servicio en segundo plano. De manera automática cuando este servicio actualice los Tweets, se actualizará la pantalla con los nuevos Tweets.

Al acceder a la opción Update Status, accederemos a la siguiente pantalla:

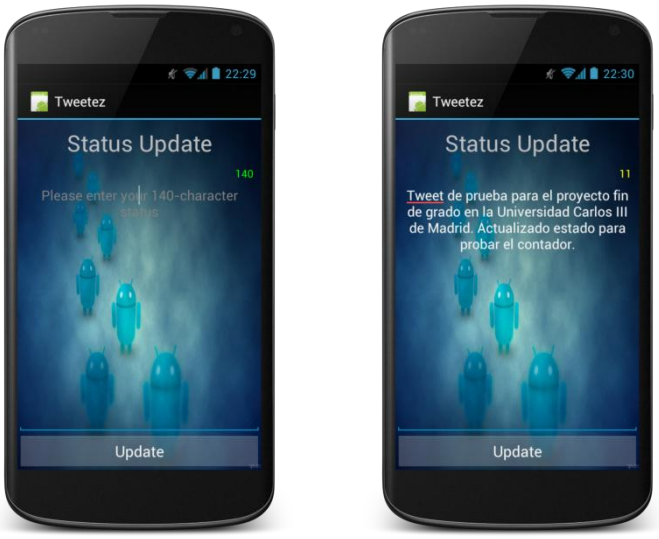

Ilustración 55 Screenshot 9 y 10

Al escribir un texto el contador cambiará según escribimos y al pulsar el botón Update, el estado escrito será enviado a nuestra cuenta para mostrarse a los seguidores.

Por último, al seleccionar la opción Statistics accederemos a la siguiente pantalla:

Ilustración 56 Screenshot 11 y 12